Determining Influence in Social Networks Using Social Capital

A Thesis
SUBMITTED TO THE FACULTY OF
UNIVERSITY OF MINNESOTA
BY

Dhruv Sharma

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
MASTER OF SCIENCE

JAIDEEP SRIVASTAVA

May 2014

Acknowledgements

I would like to thank Prof. Srivastava first and foremost for being my mentor. Providing immaculate guidance and immense support at both professional and personal fronts. I have learnt many a lesson from him, which I will remember for life. I would also like to thank him for the tremendous opportunities he has given me to enhance my research potential and professional skillset. The impact he has had on me is substantial and I will strive to keep learning from him in the future too.

I would also like to thank Karthik for having significant impact on my research and guiding me at crucial junctures. Also, I would like to thank Komal, Atanu, Chandrima, Zoheb, Ayush and all the other DMR lab members for being excellent colleagues, critiques and excellent friends. Furthermore, I would especially like to thank Kushal, Dr. Jehwan, Ankit for being friends par excellence and being there through the highs and lows of graduate life. The journey would have never been so much fun without them.

Moreover I would also like to thank my mom, Vrinda, nana sahib, nani sahib and Bhavna for being there for me during difficult times and giving me enormous support throughout this journey. I couldn’t have done any of this without them.

Last but not the least I would like to thank my dad for being there for me, guiding and supporting me throughout life. The influence he has had on me is unparalleled. He has given me tremendous amounts of input for all my research work and other graduate schoolwork, which has made a world of difference to the quality of my work.

Dedication

To Ma, Papa, Vrinda and rest of my family and friends without whose blessings and support I couldn't have done this.

Abstract

The proliferation of online social networks enables the influence of a person or an event to propagate to every corner of the globe in a very short duration of time. The problem of identifying such key sources of influence is important for a wide variety of applications from sales and marketing to public health and policies.

Most of the existing methods for identifying influencers use the process of information diffusion to discover the nodes (people) with the maximum expected information spread. In this work we have developed a novel method for identifying key influencers in a given network. This method works on the premise that people generate more value for their work by collaborating with peers having high influence. The social value generated through such collaborations denotes the notion of individual social capital. At the core of this method we use the popular valuation-allocation approach for finding the individual social capital value. In this approach first we determine the value of the entire network using a valuation function and then we do a fair allocation of this entire network's value amongst the participating nodes (people).

We show that our Valuation and Allocation functions satisfy several axioms of fairness and fall under the Myerson's allocation rule class.

Also, we implement our allocation rule using an efficient algorithm and show that our algorithm outperforms the baselines in several real life datasets. Especially, for the DBLP collaboration network our algorithm outperforms PageRank, PMIA and Weighted Degree baselines by up to 8% in terms of precision recall and F1-measure.

Furthermore, we use Hypergraphs as a tool to model group collaborations more effectively and empirically show the superiority of hypergraph edge weights as compared to dyadic edge weights for identifying influencers.

To conclude with we discuss a couple of popular distributed programming paradigms, namely MapReduce and BSP (Bulk Synchronous Parallel) and the implementation of the algorithm on these.

Table of Contents

List of Tables

List of Figures

List of Algorithms

Chapter 1

Introduction

The popularity of online social networking applications and mobile technology has been observing a staggering growth in terms of the user base over the past few years. Such a proliferation of online social networks and mobile devices enables the influence of a person or an event to propagate to every corner of the globe in a very short duration of time. Thus, the problem of identifying such people who are key sources of influence is becoming increasingly important for a wide array of applications ranging from sales and marketing [1] to public health and policies [2][3].

The field of identifying such key influencers has received a fair share interest from various research communities. Most of the existing approaches towards solving this problem [4], [5], [6], [7], [8] try to model social influence through the process of information diffusion. Such methods assume that the more influential

a user is, wider would the spread of information be through him/her. This information flow is modeled using a network (graph) structure with static or dynamic edge probabilities that are estimated from the past observation of flow of information through these edges. The notion of influence models used in these papers is that each node independently infects (influences) its neighbors with a certain probability. Then every influenced node further cascades this infection to its neighbors in the network. In a parallel thread work [9[has been done to find significant sequence cascades and these significant cascades are further used to find the top influencers specific to a context such as movies or sports.

Most of the current work captures the process of influence from a node-to-node perspective. However such methods fail to offer insights into influence in the context of the entire network, or fail when there are only a handful of observations of information flow through the network. Let us consider the following example to clarify the above points. Let us consider the following scenario. A newly appointed CEO of a company may only have a few connections and only a limited number of information flows through the organizational network. But, when we ask the question that weather he/she can

influence a new technology in the company? The answer naturally would be ‘yes’. The reason for this influence is not because of the few connections or limited information flows, but because of the control he/she exerts on the network resources (in this case all the employees of the company). Such aspects of influence cannot be captured when we only consider local interactions between two nodes. Rather to study and capture these aspects one needs to determine the value each node contributes to and derives from the entire network. We hypothesize that nodes that have high social value in the network tend to be highly influential in the network, as the case with the new CEO in our example.

For this purpose we first characterize the value of the entire network using *social capital*. There are various definitions of social capital given in the literature [10], [11], [12]. We consider the most popular one, which states *“social capital is about the value of a social network, bonding similar people and bridging diverse people, with norms of reciprocity”* [11]. This notion of social capital comprises of bridging capital that is the ability to connect a diverse set of people and bonding capital that is the ability to calibrate similar people with each other. The ability of these bonding and bridging nodes to cooperate and communicate with each

other generates an inherent value for the entire network and the overall value generated by such cooperation and communication is called *social capital of the network*.

Once, we compute the overall value for the entire network we need to allocate this value fairly amongst the participating nodes (people). And, this allocated value is the social capital value of the individual. We hypothesize that this value is proportional to the influence the individual has on the network.

Our approach uses the classical valuation-allocation framework [13] to compute the social capital value of the network and it's distribution amongst the individuals. Our valuation and allocation functions satisfy certain desired properties, which makes them fair [14]. We have devised two extremely efficient algorithms to implement the allocation rule in a weighted as well as an un-weighted setting. This algorithm uses the inherent sparse nature of social graphs to compute fractional contributions of nodes in shortest paths.

Also, we have used Hypergraphs as a tool for modeling networks, which have an inherent group structure present within them. For example an academic collaboration

network where more than two people collaborate to publish an article.

We performed a careful analysis of the effectiveness and efficiency our method against three popular baselines Weighted Degree, Page Rank and PMIA using two real world collaborative networks and observed significant improvements given by our approach against the baselines. Also, we empirically observed an improvement in the evaluation measures obtained by using weighted hypergraph modeling instead of simple graph modeling for both the networks, which were used for evaluation.

1.1 Contributions

This section highlights the various contributions this work has made.

- We have developed a novel approach for solving the problem of identifying key influencers in a social network using the notion of social capital. We merge interesting concepts from Social Science, Game Theory and Computer Science in order to develop this approach and it is different from the existing

methods, which are either centrality based or cascade based like the baselines that we have used.

- Our approach is based on the popular value-allocation model [13] for finding individual social capital value. First we compute the social capital value generated by multiple collaborations and allocate the fair share of this value amongst the nodes involved in these collaborations. We show that our allocation function falls in the class of Myerson's allocation function [14] and satisfies all the axioms of fairness.

- We have devised and implemented two extremely efficient algorithms for computing the allocation rule for social networks. These algorithms have an implicit assumption that the networks it is to be applied to are sparse. We show that our algorithm outperforms three baselines Weighted Degree, Page Rank and PMIA. Especially, in the DBLP dataset our method has *8%* higher values for precision, recall and F_1-Measure.

- We used hypergraphs as a tool to model higher order relationships inherent within group networks. As a part of this work we explore different weighing

schemes for hypergraphs and empirically determine the most appropriate one for this context. Finally, we show that modeling higher order relationships using hypergraphs improved the performance of our algorithm as compared to the simple graph modeling approach.

1.2 Organization

Rest of this manuscript is organized as follows. Chapter 2 describes the various related work that we surveyed in order to come up with our approach. Chapter 3 describes the proposed approach in detail. It lists the used valuation and allocation functions, and contains the proofs for the axioms of fairness (properties) satisfied by these functions. Chapter 4 discusses the simpler un-weighted algorithm we have devised to compute the allocation function for a given un-weighted social network. Chapter 5 discusses the more complex algorithm which computes the allocation function for a weighted social network, in general social networks are weighted as the strength of ties between people vary greatly. Chapter 6 discusses the various weighing schemes for graphs and hypergraphs, in this chapter we also determine the most appropriate hypergraph weighing scheme for this context and provide an algorithm to convert a hypergraph

to a graph with minimum information loss (so that simple graph algorithms can work on top of it). Chapter 7 discusses the details of the experimentation we have done in order to compare and contrast our algorithm against the baselines, as well compare the graph and hypergraph weighed variants. This chapter contains extensive details about the datasets, parameters, evaluation measures and infrastructure used for out experiments. Chapter 8 discusses a couple of popular distributed frameworks that could be used to implement our approach for extremely large-scale graphs. To conclude with chapter 9 lists the conclusions and possible future directions of this work.

Chapter 2

Related Work

2.1 Introduction

This chapter discusses the various other works, which have been used for the deeper level of understanding of the various topics involved in the research presented in this document.

This chapter is organized into four separate sections, each dealing with a particular field which has been a part of this research. The second section, section 2.2 talks about the influence maximization problem and some solution approaches, which have typically been used to find influential nodes in a network. The third section, section 2.3 talks about the various aspects of co-operative game theory, which have been used to model our approach towards finding the value of each node in the network in a fair manner. The fourth section, section 2.4 talks about the notion of social capital and how

different people have defined it from a social science perspective. The fourth and final section, section 2.5 sheds light upon the usage of Hypergraphs as modeling tools for capturing higher order relationships.

2.2 Influence Maximization

The problem of finding influencers in a given social network (graph) is often studied and solved as an influence maximization problem [4], [5], [6], [7], [8], [15]. The influence maximization problem can be defined as the task of identifying *k* number of nodes (often referred to as top-k nodes) such that the average influence spread achieved via these nodes is maximal under a specified influence propagation model.

There are two popular influence propagation models that are present in the literature, one is the Independent Cascade (IC) model and the other is the Linear Threshold (LT) model [4]. In the Independent Cascade model each node (person) that is influenced at time point *t-1* has a single chance to influence a given neighbor with a certain probability at time point *t*. While, the Linear Threshold model assumes that each node (person) can influence it's neighbor with certain probability, but the neighbor only gets influenced if its threshold of getting

influenced is crossed due to a combined effect of its neighbors at any time point t. Both these models assume that we are given an edge influence propagation probability model, that is each edge has a probability associated with itself, which is the probability of influence flow through that edge. The most popular choices for determining these edge propagation probabilities are the weighted cascade model [4] and the trivalency model [7]. In the weighted cascade model the probability of propagation along a directed edge u to v is $\frac{1}{in\_degree(v)}$. While, in the trivalency model the probability is selected uniformly at random from a given set of three probability values. Typically, these values are *{0.1, 0.01, 0.001}* corresponding to high, medium and low influencers.

Most of these influence maximization approaches [4], [8], [6] use Monte Carlo (MC) simulation technique to find the top-k nodes in the network that maximize the influence spread in the network on an average. These MC simulations are necessary since these are probabilistic models. Some other recent work [6], [8] focuses on optimizing the greedy heuristics of MC simulations. In [6] the authors proposed an optimization strategy CELF using the sub-modularity property of the influence maximization

function. In [8] the authors proposed a shortest path based model and describe an efficient algorithm for computing the influence spread using this model. There have been some recent approaches [9] that extract the frequent paths of content percolation from the underlying data. The frequent flow paths extracted are then modeled as a cascade structure over which a greedy sub-modularity maximization procedure is applied to extract the top-k influencers. All these models use an underlying influence propagation model and use efficient techniques for MC simulations to identify influencers.

2.3 Game Theory

In the realm of game theory there are a plethora of papers that discuss network formation, and the efficiency and stability of the formed networks [13], [16], [17], [18]. The seminal work of Jackson and Wolinsky [13] defines a valuation function for a network and an allocation function that distributes this value amongst the nodes participating in this network. Their model does not take into account the bridging benefits, which a node receives. In [16] a non co-operative game model is proposed to study network formation and they assume that benefits do not decay due to non-neighbor nodes as they are further away from the given node. On similar lines

authors in [17] propose a non co-operative game model that assumes that bridging benefits exist only for paths that have length of two. This corresponds to a node bridging two of its neighbors and getting benefits for that bridge, all other bridging benefits are ignored in this work. In a more recent work [18] authors formulate the problem of network formation using several important properties such as cost of maintaining links and decaying benefits with distance. These works discuss fair division of value amongst nodes in a network. But, their focus is to understand the stability and efficiency of network formation. Whereas, the purpose of this work is to propose a method that determines the (social capital) value of nodes (people) in a given network, and through that determine the top influencers in the network.

2.4 Social Capital

There have been recent attempts [19], [20], [21] focused around defining the notion of social capital. In [19] and [21] authors discuss bonding and bridging capital as the two components of social capital and uses the concept of implicit affinities and explicit affinities between nodes to compute the bridging and bonding capitals. This model assumes that bonding and bridging capitals are only attributed by the immediate neighbors and ignore non-

neighbor benefits. They also aim to compute the social capital value of each node but they ignore the benefits which can be yielded due to non-neighbor nodes which exist on longer path lengths. However, the social capital is generated because social interactions of all nodes in the network both direct and indirect. Therefore, in this work we focus on computing the overall value of such direct and indirect interactions and the fair allocation of this value amongst the participating nodes (people).

2.5 Hypergraphs

Primer lessons on Hypergraphs can be found at [26]. Hypergraph representation has been applied in a variety of domains such as Biology [35], Databases [36] and Data Mining [37]. There has been an increase in the interest to use hypergraphs towards understanding complex networks [27]. Authors in [38] describe and discuss clustering, classification and embedding problems in hypergraphs. Authors in [39] proposed models of preferential attachment for hypernetworks. In [31] authors looked at the problem of dynamic shortest path computation for hypergraphs that had weighted hyperedges and used these distances to compute closeness centrality for the Enron e-mail dataset. With the increasing availability of

large-scale group networks we see hypergraphs as an essential tool for modeling group interactions.

Chapter 3

Proposed Approach

3.1 Introduction

In this chapter we discuss in detail the problem formulation for identifying the key influencers in a network and the approach we have taken to solve it. From a very high level perspective the problem can be visualized as that of assigning a value, which may be referred as the social capital, to each node of a given collaborative network, in a fair manner. We believe the higher is the social capital of a node the greater is the influence of the node in network.

The approach we have taken to compute the social capital is based on the popular Valuation-Allocation framework in which first the value of the entire graph is determined using a Valuation function and then fairly divided amongst the constituent nodes via an Allocation function. Furthermore, in this section we establish that the

Valuation and Allocation functions that we have devised fall under the class of Myerson Allocation rules [14] and satisfy all the required axioms of fairness.

This chapter is organized as follows. Section 3.2 introduces the notion of Social Capital; Section 3.3 captures the overall approach used to solve the problem of fairly computing the Social Capital Value for each participating node in a network. Furthermore, section 3.3 and section 3.4 describe the proposed Valuation and Allocation functions and prove that they fall in the Myerson's Allocation rules class.

3.2 Notion Of Social Capital

Amongst the various definitions that exist in the literature for capturing the concept of Social Capital [10][11][12], arguably one of the most popular one verbatim is "Social Capital is about the value of social networks, *bonding* similar people and *bridging* diverse people, with norms of reciprocity" [11]. The notion of Social Capital includes both the bonding and bridging capital.

Bonding capital is the ability to calibrate similar people against each other, and bridging capital is the

ability to connect diverse people. A small example to illustrate the concept of bonding and bridging capital is given in Figure 3.1. There we can see that the bonding capital of nodes A and G is the highest in the network as each of them keeps three nodes together. Whereas, node B has the highest bridging capital since it acts as a bridge between two hubs in the network, and hence is extremely important for information exchange.

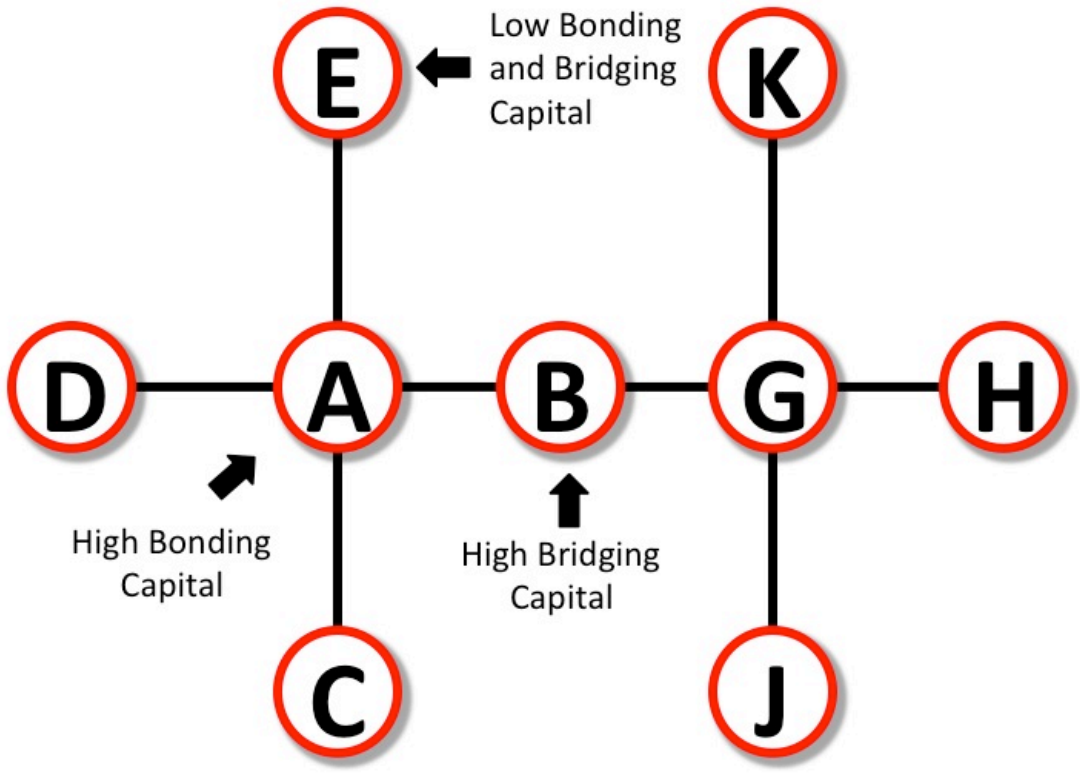


Figure 3.1: An illustrative example of bridging and bonding capital

The ability of these bonding and bridging nodes to cooperate and communicate with each other creates an inherent value for the entire network. For example, in an academic co-authorship network interdisciplinary

researchers can be considered people with strong bridging capital and research leaders in their own community can be thought of as people with a high bonding capital. Of course, there can be people with both high bonding and bridging capitals both. The overall value generated by these collaborations is termed as the Social Capital of the entire network.

3.3 Description of the Approach

Now that we have established the notion of Social Capital for a network, we have to work towards a mathematical model that will assign a value to the Social Capital for a network, and will divide this value amongst the participating nodes in a fair manner. We achieve the entire process using the popular Valuation-Allocation function method.

The first step is using the Valuation function (which satisfies a certain set of properties) defined in section 3.4 to compute the Social Capital value for the entire network. The next step is to use the Allocation function defined in section 3.5 for dividing the calculated Social Capital value of the network amongst the participating nodes. The process is illustrated in Figure 3.2. After computing the fair share of social capital for each node,

we hypothesize that this value is proportional to the potential of a node to influence the network.

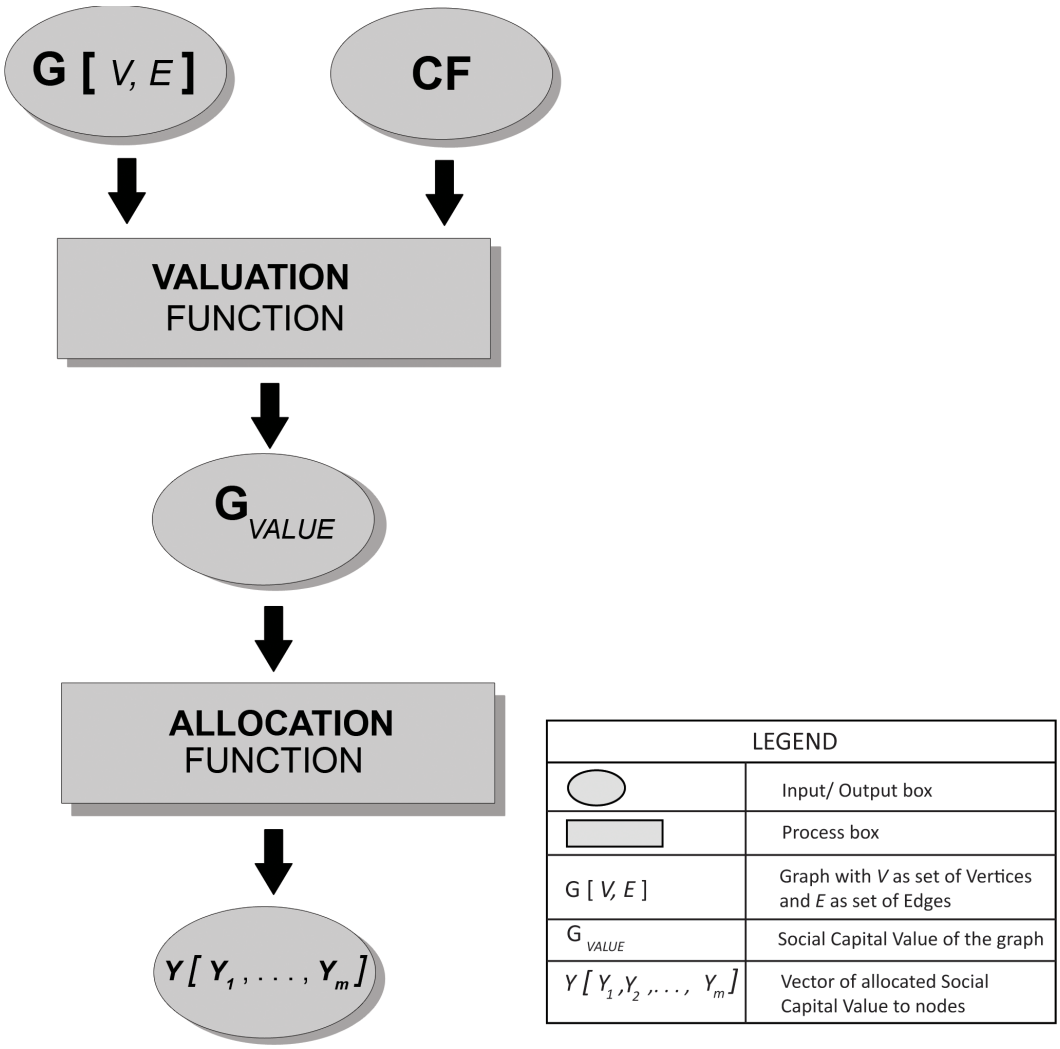


Figure 3.2: Flow diagram of the overall approach of computing Social Capital value for the nodes in the given network

3.4 Valuation Function

This section describes the proposed Valuation function (for weighted as well as un-weighted graphs) and

discusses in detail the various desirable properties it needs to satisfy.

3.4.1 Proposed Valuation Function

Let $g = (V,E)$ be a graph with vertex set V and edge set E. We have devised a valuation function v(g) that can capture the definition of the social capital for a network aptly. The main points that had to be kept in mind while defining this function were that both the bonding capital value and the bridging capital values should be captured. For a given graph g, the valuation function v(g) can be defined as follows,

$$v(g) = \sum_{i \in V, j \in V, i \neq j} b\left(d_g(i,j)\right)$$

Here in (3.1), the distance function d_g represents the distance of the shortest path between the nodes i and j belonging to the network. Here we assume that people (nodes) often make new connections to reach newer friends through shorter paths. This assumption is also consistent with many network formation studies [13], [22]. For a weighted graph the distance 'd' between the nodes depends on the weights of the edges and can be any positive real number, and for an un-weighted graph this distance

depends on the path length and can be any positive integer.

We call the function $b(d)$ the benefit function. This function returns the contribution of a shortest path of distance d. This function can be chosen as per the given requirements. Hence v(g) actually defines a class of valuation functions, and one can define a specific valuation function by choosing an appropriate benefit function. For our setting that is social in nature we chose this function to be of exponential decay in nature. The chosen function can be mathematically represented as the following.

$$b(d) = \begin{cases} e^{-\lambda d}, & if\ hops(i,j) \leq hops_{\max} \\ 0, & otherwise \end{cases} \qquad (3.2)$$

Here, d is the distance of the shortest path, and $hops(i,j)$ is the number of hops (un-weighted edges) in the given shortest path. One can use $hops_{max}$ as a control parameter to eliminate longer paths from contributing to the valuation function. This would help us save significant computational time for large-scale graphs. If we want to consider all the shortest paths in the graph to contribute towards the evaluation function, then we can

simply specify the $hops_{max}$ same as the un-weighted diameter of the graph (or more).

It is noteworthy here that when there is no path between two nodes i and j which is lesser than or equal to the specified hop_{max}, the distance d between them becomes ∞ (infinity) and the corresponding benefit value becomes 0.

This evaluation function captures the two essential properties required by our evaluation function. (1) Benefits due to immediate neighbors and (2) decaying benefits from non-immediate neighbors. Benefits due to immediate neighbors are captured by paths, which only have a single hop between the source and the destination nodes (paths of hop length = 1 in weighted graph and paths with distance 1 for un-weighted graph). Similarly, paths, which have more than one hop between the source and the destination nodes, capture benefits due to non-neighbor nodes and this is made exponentially decaying with the distance of the path, due to the choice of the benefit function specified in equation (3.2). So, for a completely connected un-weighted graph where each node is connected to the other by a shortest path distance of 1, the social capital value will be maximum and it's value is $v(g) = \frac{b(1) \times |V| \times (|V|-1)}{2}$. In contrast, a graph with no edges

will have the minimum possible social capital value of $v(g) = 0$.

The immediate neighbor benefits measured by the proposed valuation function eventually capture the bonding capability of the network and the non-immediate benefits capture the bridging capability of the network. Thus, our valuation function captures both the aspects of Social Capital and is an ideal fit for computing it.

The proposed Valuation function needs to satisfy a couple of properties as well to be acceptable within the proposed framework. Section 3.4.2 specifically talk about there properties.

3.4.2 Properties Of Valuation Function

In this section we discuss the desired properties for the proposed Valuation function and also show theoretical proofs that the proposed function satisfies all of them. The desired properties are (1) Anonymity and (2) Component Balance. The following subsections discuss each of these in detail.

3.4.2.1 Anonymity

This section discusses in detail about the Anonymity property and gives a proof that the proposed Valuation function satisfies this desired property.

Definition: *Given a permutation π of the set of nodes N, and any given graph $g \in \psi$, let the permuted graph be $g^{\pi} = \{ (\pi(i), \pi(j)) \,|\, (i,j) \in g \}$. Then a valuation function v is said to be anonymous if $v(g^{\pi}) = v(g)$.*

Note that graph g and graph g^{π} share the same network structure, only the nodes have new labels. Hence, an anonymous valuation function v is independent of labels of the nodes.

Lemma 1. *The proposed valuation function v(g) satisfies Anonymity property.*

Proof. By definition of the valuation function *v(g)* we know,

$$v(g^{\pi}) = \sum_{\pi(i) \in V^{\pi}, \pi(j) \in V^{\pi}, \pi(i) \neq \pi(j)} b\left(d_{\mathrm{g}}(\pi(i), \pi(j))\right)$$

Since, there is a one-to-one mapping of all nodes of g to those of g^{π}. Hence, there is no change in the number of

shortest paths in the graph g^{π} as compared to that of graph g and the above equation can be re-written as following.

$$v(g^{\pi}) = \sum_{i \in V, j \in V, i \neq j} b\left(d_{\mathrm{g}}(i,j)\right)$$

$$= v(g)$$

Hence, the Anonymity property has been proved for the proposed Valuation function.

3.4.2.2 Component Balance

This section discusses in detail about the Component Balance property and gives a proof that the proposed Valuation function satisfies this desired property.

Definition: *Let* $\Omega(N) = \{C_1, C_2, \ldots, C_k\}$ *be the set of all connected components of graph* $g \in \psi$. *A valuation function* v *is component additive, if it satisfies* $\sum_i V(C_i) = v(g)$.

This property states that the sum of the valuation function of a graph is equal to the sum of valuation function of the components. Hence, it says that all the connected components are separate entities in themselves

and their valuation function is independent of the other components, which are disconnected from them.

Lemma 2. *The proposed valuation function v(g) satisfies Component Balance property.*

Proof. Let V_p be the set of vertices belonging to the component C_p. Then,

$$v(C_p) = \sum_{i \in V_p, j \in V_p, i \neq j} b(d_g(i,j))$$

When we take the sum of the valuation function v of all the connected components of g. We get,

$$\sum_{p=1}^{k} v(C_p) = \sum_{p=1}^{k} \sum_{i \in V_p, j \in V_p, i \neq j} b(d_g(i,j))$$

The above equation can be rewritten as:

$$= \sum_{i \in \cup V_p, j \in \cup V_p, i \neq j} b(d_g(i,j))$$

Since the set of vertices and edges within each connected component are unique and disjoint from any other component. Hence, each vertex will only be connected to other vertices in it's own connected component and not to any vertex outside the connected component it exists in.

Each connected components shortest paths are independent of the other components. Therefore, all shortest paths are counted exactly once. The above equation can be written as:

$$= \sum_{i \in V, j \in V, i \neq j} b\left(d_{\mathrm{g}}(i,j)\right)$$

$$= v(g)$$

Hence, the Component Balance property has been proved for the proposed Valuation function.

3.5 Allocation Function

In this section we discuss in detail about the proposed allocation function and the various axioms of fairness shown in [23] it has to satisfy in order to fall into the class of Myerson Value allocation [14].

3.5.1 Proposed Allocation Function

An allocation function is defined as $Y: \psi \longrightarrow \Re^n$, where $Y = [Y_1, Y_2, \ldots, Y_n]$ represents the Social Capital value assigned to nodes 1 through n.

Our proposed Allocation function is based on the idea that each node contributes a certain value to the network by existing in shortest paths in the network (these paths can be of minimum 1 hop and maximum number of hops can be equal to the diameter of the graph). The diameter can be a maximum of $|V| - 1$. We measure the fractional contribution of each node $k \in V$ for all the shortest paths it participates in. For a reduction in computational time, instead of all shortest paths in the network, we restrict ourselves to paths which have hops less than or equal to $hops_{max}$ (which is specified as a control parameter to the method). This pruning of shortest paths is also performed while computing the Social Capital of the entire graph using the valuation function.

For a given graph g the fractional contribution all paths of distance d and hops h towards a node $k \in V$ can be defined as following:

$$\alpha_k^{d,h}(g) = \frac{\beta_k^{d,h}}{h+1}$$

Where, $\beta_k^{d,h}$ is the total number of paths of distance d and hops h on which node k is present. The denominator in the above equation ensures that the total value of the path (having distance d) is equally distributed amongst all

the nodes, which are present on that path. It can be easily deduced that the total number of nodes on a path having h hops is $(h+1)$.

This can be simplified if the given graph is un-weighted since in an un-weighted graph the distance and hops are the same. The fractional contribution for un-weighted graphs can be simply be written as following.

$$\alpha_k^h(g) = \frac{\beta_k^h}{h+1}$$

We will use the more general weighted definition of the Fractional contribution further in this chapter.

Formally the proposed Allocation function can be defined as $Y: \psi \longrightarrow \Re^n$, where the k^{th} component of Y i.e. $Y_k(g)$ is the sum of the benefit function values weighted by the corresponding fractional contribution for the node k for all possible combinations of shortest path distances d and hops h. Formally the proposed allocation function is given in the equation below.

$$Y_k(g) = \sum_{h=1}^{|V|-1} \alpha_k^{d,h}(g) * b(d)$$

Note that a finite graph g will have an enumerable number of shortest paths, hence the set of all shortest distances in the graph will also be enumerable. In the above equation we have assumed that the set of all shortest distances in graph g is {d1, d2, d3, ……., dm}.

3.5.2 Properties Of The Allocation Function

When a set of players $N = \{1,\ldots,n\}$ cooperate with each other to achieve a common goal, the profit yielded must be shared amongst the players in a fair manner. Shapely [24] proposed a mechanism, called the *Shapely Value* function to achieve this fair division, considering several axioms of fairness. However, Shapely assumed that there is complete cooperation between all the players in the game, which is not the scenario in most of the practical cases. Myerson [14] proposed a new mechanism for the fair division of profits amongst the players when the cooperation is defined by a graph structure that is not completely connected. The fair share of value received by each player as a result of this mechanism is popularly referred to as *Myerson Value*. Any function that satisfies four properties; Anonymity, Component Balance, Improvement property and Weak Link Symmetry falls in the class of Myerson's value functions [23]. In this section

we define these properties and show that the proposed allocation function satisfies all of them.

3.5.2.1 Anonymity

This subsection defines the Anonymity property for an allocation function and proves that the proposed allocation function satisfies this.

Definition: *Given a permutation π of the set of nodes N, and any given graph $g \in \psi$, let the permuted graph be $g^{\pi} = \{ (\pi(i), \pi(j)) \,|\, (i,j) \in g \}$. Then an allocation function Y is said to be anonymous if $Y_{\pi(i)}(g^{\pi}) = Y_i(g)$.*

The Anonymity property ensures that the allocation function is independent of the node labels and remains the same even if the labels of the nodes are jumbled.

Lemma 3. *The proposed Allocation function Y(g) satisfies Anonymity property.*

Proof: By the definition of the allocation function we know.

$$Y_{\pi(k)}(g^{\pi}) = \sum_{h=1}^{|V|-1} \alpha_{\pi(k)}^{d,h}(g^{\pi}) * b(d)$$

Now, we know that the permutation g^{π} contains an exact one-to-one mapping to the labels of g, and hence there is no difference in the graph structure. Using this one-to-one mapping of the node labels the above equation can be written as following.

$$= \sum_{h=1}^{|V|-1} \alpha_k^{d,h}(g) * b(d)$$

$$= Y_k(g)$$

Hence, it has been proved that the proposed allocation function satisfies the Anonymity property.

3.5.2.2 Component Balance

This subsection defines the Component Balance property for an allocation function and proves that the proposed allocation function satisfies this.

Definition: Let the set of all connected components of graph g be $\Omega(\mathrm{N}) = \{C_1, C_2, \ldots, C_k\}$, then the allocation function Y is component balanced if $\sum_{j \in C_i} Y_j(g) = v(C_i), \forall\, C_i \in \Omega(N)$.

The component balance property for the allocation function ensures that the value within a connected

component C_j is only allocated to the nodes within that component. The value of one component should not affect that of another.

Lemma 4. *The proposed Allocation function Y(g) satisfies Component Balance property.*

Proof: Let V_k be the set of nodes of component C_k and $\gamma_k^{d,h}$ be the total number of shortest paths of distance d and hops h in component C_k. The shortest path length between nodes i and j in component C_k is $d_k(I,j)$. Then,

$$\sum_{j \in V_k} Y_j(C_k) = \sum_{j \in V_k} \sum_{h=1}^{|V_k-1|} \alpha_j^{d,h}(C_k) b(d)$$

$$= \sum_{j \in V_k} \sum_{h=1}^{|V_k-1|} \frac{\beta_j^{d,h}}{h+1} b(d)$$

$$= \sum_{h=1}^{|V_k-1|} \frac{b(d)}{h+1} \sum_{j \in v_k} \beta_j^{d,h}$$

$$= \sum_{h=1}^{|V_k-1|} \frac{b(d)}{h+1} \gamma_k^{d,h} (\mathrm{h}+1)$$

$$= \sum_{i \in V_k, j \in V_k} b\left(d_g(i,j)\right)$$

$$= v(C_k)$$

Hence, it is proved that the proposed allocation function satisfies the Component Balance property.

3.5.2.3 Improvement Property

This subsection defines the Improvement property for an allocation function and proves that the proposed allocation function satisfies this.

Definition: An allocation rule Y satisfies improvement property if $Y_z(g \cup \{e\}) > Y_z(g), \forall e = (i,j) \notin g,\ z \in N/\{i,j\}$, then $Y_i(g \cup \{e\}) > Y_i(g)\ and\ Y_j(g \cup \{e\}) > Y_j(g)$ must be satisfied.

This properties implies that while adding a new edge *e (i,j)* to the graph *g*, if the utility of any other node apart from i or j in the graph increases then the utility of either i or j must increase.

Lemma 5. *The proposed Allocation function Y(g) satisfies Improvement property.*

Proof: If edge *e = (i,j)* is added and $Y_z(g \cup e) > Y_z(g)$ implies that the node *z* is in the new shortest path *P* =

(s,…, z,…,i, j,…, t) of distance *d* due to the addition of edge *e* to graph *g*. The allocation Y_z of this node can increase if $\alpha_z^{d',l'} > \alpha_z^{d,l}$. As a consequence $\beta_z^{d',l'} > \beta_z^{d,l}$. This is true for any node that lies in the shortest path *P*, and hence without loss of generality we can say that $\beta_z^{d',l'} > \beta_z^{d,l}$ and therefore $Y_i(g \cup e) > Y_i(g)$. This proves the improvement property for the given allocation function.

3.5.2.4 Weak Link Symmetry

This subsection defines the weak link symmetry property for an allocation function and proves that the proposed allocation function satisfies this.

Definition: An allocation rule *Y* satisfies Weak Link Symmetry if $Y_i(g \cup \{e\}) > Y_i(g)$, then $Y_j(g \cup \{e\}) > Y_j(g)$ must hold for all $e = \{i,j\} \notin g$.

This is a more general form of equality criterion specified by Myerson in [14]. We prefer to ensure this criterion because the utility received by adding a new edge in the graph may not be necessarily due to equal contributions from both the nodes.

Lemma 6. *The proposed Allocation function Y(g) satisfies weak link symmetry property.*

Proof: By adding a new edge *e = (i,j)* to the graph *g*. If $Y_i(g \cup e) > Y_i(g)$ then there must be a new shortest path *P = {s, …, i, j, …, t}* from some node *s* to some other node *t* passing through the edge *e (i,j)* of distance *d*. and hops *h*. Then, $\alpha_i^{d',h'} > \alpha_i^{d,h}$, now since the new shortest path has to go through the edge *e* node *j* must be a part of this new shortest path. Hence, the corresponding $\alpha_j^{d',h'} > \alpha_j^{d,h}$. Due to the increase in this fractional contribution we can conclude that If $Y_j(g \cup e) > Y_j(g)$. This proves the weak link symmetry property for the given allocation function.

Chapter 4

Un-Weighted Algorithm

4.1 Introduction

This chapter provides conceptual level details of the algorithm that we have used to implement the valuation-allocation approach for un-weighted sparse graphs. The algorithm needs to efficiently determine all the shortest paths in the graph between all pairs of nodes and distribute the contribution of a shortest path fairly amongst all the nodes lying on the path. The algorithm consists of two phases namely the forward propagation phase and the backward propagation phase. Forward propagation phase determines all shortest paths from a source node to all the other nodes (target node), and maintains with each target mode the length of the shortest path to it along with the count of the number of shortest paths. The backward propagation phase distributes credits for a shortest path to all the nodes existing in the shortest paths proportionally. The

algorithm also takes a control parameter that allows restricting the maximum path lengths to be considered for valuation-allocation functions. The exact flow of the algorithm is also described using an illustrative example. Finally, an analysis of the computational complexity of the algorithm is provided.

The chapter is organized as follows. Section 4.2 defines the set of notations used for the un-weighted algorithm. Section 4.3 describes the algorithm. Section 4.4 gives an example to illustrate the algorithm. To conclude with section 4.5 discusses the theoretical complexity of the algorithm.

4.2 Notations for an Un-weighted Graph

The notations used in this chapter are slightly simplified as compared to the ones used in Chapter 3 since it was describing the more generalized valuation and allocation functions, which also incorporated edge weights. In this chapter the distance d and hops h are the same because each edge has a weight 1.

α_k^l , denotes the fractional contribution of paths of length l to node k. In chapter 3 this was a more complex

notation $\alpha_k^{d,h}$ which denoted the fractional contribution of paths of distance *d* and hops *h* to node *k*. But, in the un-weighted version the hops and distance are the same.

L, denotes the maximum path length (hops) to consider for computing the allocation function. In chapter 3 this notation was $hops_{max}$ but since path length and hops are same in this context we simplified the notation.

4.3 The Algorithm

Our algorithm *NESCap*, as listed below, implements the allocation rules described in the previous section. The core part of the algorithm *NESCap* is encapsulated in the sub algorithm *ComputeFC*, which computes α_k^l , where α_k^l is the fractional contribution made to node *k* by all shortest paths of length *l* that contain the node *k*. Once α_k^l gets computed for all $k \epsilon V$ and for all path lengths *l* varying from *1* to $|V|-1$, the algorithm *NESCap* simply sums up the weighted fractional contributions for each node *k* and stores it in *Y[k]*, with the benefit function *b(l)* serving as the weight. Finally, a vector *Y = […]* which is a vector containing the weighted fractional contributions for each node is returned as the output by *NESCap*.

As the benefits due to path length *l* decreases with the increase in *l*, the contribution to social capital value by path lengths beyond a limit may become insignificant. Hence algorithm *NESCap* allows the user to provide the limit *L* ($hops_{max}$) on the maximum path length to be considered during computation instead of the diameter of the graph. Choosing lower values of *L* significantly improves the computational performance of the algorithm.

A simple approach to compute α_k^l for all values of *l* and *k* would be to find out all shortest paths between all pairs of vertices in the graph *g*, and then count how many times each node *k* has appeared on various shortest paths of length *l* varying from *1* to *L*. The algorithm for computing all shortest paths between all pairs of vertices has the time complexity of $O(|V|^3)$, which is not acceptable for the graph sizes we are generally going to deal with. Hence we developed *ComputeFC* as a modified version of Brandes betweenness centrality computation algorithm [4]. The Brandes algorithm computes betweenness centrality for each node *v* in the graph *g*, which is the fraction of the number of shortest paths in which the node *v* is present to the total number of shortest paths in the graph *g*. The complexity of Brandes algorithm is $O(|V||E|)$, which is a significant improvement over $O(|V|^3)$ for sparse graphs,

since in sparse graphs the number of edges in the graph $|E|$ is of linear order of the number of nodes in the graph $|V|$. The Brandes algorithm [4] however does not compute the differentiated contributions based on varying path lengths. Our algorithm *ComputeFC* calculates differentiated contributions for varying path lengths for each node k in the graph without adding significantly to the complexity of the algorithm.

The algorithm *ComputeFC* iterates over each node v in the graph g, considering v as the source node in that iteration. With v as the source node, the algorithm performs two phases- forward propagation and backward propagation. These phases are explained in the sections below.

4.3.1 Forward Propagation Phase

The lines *8* to *24* in the pseudo code correspond to the forward propagation phase of the algorithm. The forward propagation phase uses breadth first search to find the shortest path from the source node v to each of the other nodes in the graph g, assuming that all edges in graph are of length *1*. Other than just finding the lengths of these shortest paths, it also counts the number of shortest paths found to each of the nodes. In each

iteration a node is picked up from the queue to propagating packets to its adjacent nodes. The propagation begins with the source node. The packets received at a node are summed up to determine the count of the number of shortest paths to that node. If a node *u* that is at distance *d* from the source node *v* receives a total of *n* packets then it forwards those *n* packets to each of its neighbors that are at a distance of *d+1* from the source node.

4.3.2 Backward Propagation Phase

In the backward propagation phase (*lines 25-31*), the nodes get processed in the reverse order of the forward propagation. The nodes that are at distance *L* from the source node will get processed first, followed by the nodes that are distance *L-1* and so on; with the source node getting processed at the end. During backward propagation a node *u*, which was found at distance *d* from source node *v*, will back propagate packets to all its adjacent nodes from which it received packets during forward propagation (these nodes will be at distance *d-1* from the source node). Unlike forward propagation in backward propagation a node *u* back propagates multiple packet counts, with each count corresponding to an integral distance in the range *[1...L]*. In back

propagation a node splits a packet count amongst its relevant adjacent nodes in the same ratio in which it received packets from them during forward propagation.

Algorithm 1: NESCap

Input: g: social network graph; L: maximum path length
Output: $Y = [y_1, ..., y_n]$: social capital of nodes in V

$Y \leftarrow 0$;
$\alpha_v^l \leftarrow ComputeFC(g, L)$;
for $i \leftarrow 1$ **to** n **do**
    for $l \leftarrow$ **to** L **do**
        $y_i \leftarrow y_i + \alpha_i^l * b(l)$;
Return Y;

Algorithm 1: Pseudo code for the NESCap algorithm for un-weighted sparse graphs

Algorithm 2: ComputeFC

Input: g: social network graph; L: maximum path length
Output: $\alpha_v^l, \forall v, \forall l$: fractional contribution for each node for various path lengths

```
1  Initialize Queue Q;
2  Initialize Stack S;
3  for each v in V do
4      v.found ← true;
5      v.pathLength ← 0;
6      v.packets ← 1;
7      Q.enqueue(v);
8      while (!Q.empty()) do
9          u ← Q.dequeue();
10         S.push(u);
11         l ← u.pathLength + 1;
12         if l > L then
13             continue;
14         for each w adjacent to u do
15             if !w.found then
16                 w.found ← true;
17                 w.pathLength ← l;
18                 w.packets ← u.packets;
19                 α_w^l ← u.packets;
20                 Q.enqueue(w);
21             else
22                 if w.pathLength == l then
23                     w.packets ← w.packets + u.packets;
24                     α_w^l ← α_w^l + u.packets;
25     while (!S.empty()) do
26         u ← S.pop();
27         for each w adjacent to u do
28             if w.pathlength < u.pathlength then
29                 for l ← 1 to L do
30                     f ← w.packets / u.packets;
31                     α_w^l = α_w^l + α_u^l * f;
32         u.found ← false;
33         u.pathlength ← ∞ ;
34         u.packets ← 0;
35 for each v in V do
36     for l ← 1 to L do
37         α_v^l ← α_v^l / (l+1);
38 Return α_v^l, ∀v, ∀l;
```

Algorithm 2: Pseudo code for ComputeFC algorithm for un-weighted sparse graph

4.4 Illustrative Example

In this section we explain the algorithms *NESCap* (Algorithm 1) and *ComputeFC* (Algorithm 2) with an illustrative example. Consider that we are given a graph listed in Figure 4.1 and the benefit function $b(l) = e^{-1}$. The value of the entire graph can be computed as following $v(g) = \sum_{l=1}^{L} b(l) \times number\ of\ shortest\ paths\ of\ length\ l$. The given graph has *8* shortest paths of length *1=1*, *12* shortest paths of length *1=2*, *8* shortest paths of length *1=3* and *4* shortest paths of length *1=4*. Hence, the value of the graph *v(g) = 5.0386*. This calculation is shown below.

$$\boldsymbol{v(g)} = 8 \times e^{-1} + 12 \times e^{-2} + 8 \times e^{-3} + 4 \times e^{-4}$$

$$\boldsymbol{v(g)} = 5.0386$$

Let us momentarily assume that *ComputeFC* has given us fractional contributions α_i^l listed in Table 4.1. Then computing the allocated (social capital) value for each node is very straightforward. As we know that $Y_i = \sum_{l=1}^{L} \alpha_i^l \times b(l)$. Hence the value of a node can be computed by taking a weighted sum of a column of the corresponding node,

using *b(1)* as the weight. The calculations are shown below.

$$\boldsymbol{Y_1} = \frac{2}{2} \times e^{-1} + \frac{3}{3} \times e^{-2} + \frac{4}{4} \times e^{-3} + \frac{4}{5} \times e^{-4} = 0.5677$$

$$\boldsymbol{Y_2} = \frac{2}{2} \times e^{-1} + \frac{5}{3} \times e^{-2} + \frac{4}{4} \times e^{-3} + \frac{2}{5} \times e^{-4} = 0.6506$$

$$\boldsymbol{Y_3} = \frac{2}{2} \times e^{-1} + \frac{5}{3} \times e^{-2} + \frac{4}{4} \times e^{-3} + \frac{2}{5} \times e^{-4} = 0.6506$$

$$\boldsymbol{Y_4} = \frac{4}{2} \times e^{-1} + \frac{10}{3} \times e^{-2} + \frac{8}{4} \times e^{-3} + \frac{4}{5} \times e^{-4} = 1.3012$$

$$\boldsymbol{Y_5} = \frac{2}{2} \times e^{-1} + \frac{5}{3} \times e^{-2} + \frac{4}{4} \times e^{-3} + \frac{2}{5} \times e^{-4} = 0.6506$$

$$\boldsymbol{Y_6} = \frac{2}{2} \times e^{-1} + \frac{5}{3} \times e^{-2} + \frac{4}{4} \times e^{-3} + \frac{2}{5} \times e^{-4} = 0.6506$$

$$\boldsymbol{Y_7} = \frac{2}{2} \times e^{-1} + \frac{3}{3} \times e^{-2} + \frac{4}{4} \times e^{-3} + \frac{4}{5} \times e^{-4} = 0.5677$$

As we can see node *4* seems to be having the highest social value, as he tends to be in the center of the network. Nodes *2, 3, 5* and *6* have the same value as they hold symmetrical positions in the network. Similarly nodes *1* and *7* have the same value for the same reason.

Now, we will illustrate how the *ComputeFC* algorithm works using the same example. Figure 4.2 is a pictorial representation of the Forward Propagation phase of the C*omputeFC* algorithm when node *1* is taken as the source node. The forward propagation phase starts with node *1* sending a message and discovering node *2* and node *3* at a path length of 1 from the source node shown in Figure 4.2(a). In the second and third iterations of the forward propagation phase shown in Figure 4.2(b), node *2* and node *3* increment the path length by *1* and send the number of packets they received (*1)* to their adjacent nodes which have not yet been discovered, that is only node *4* (since node *1* initiated the process it's already marked as discovered), and it combines all the packets it received *(2)*. In the third iteration shown in Figure 4.2(c) node *4* the only node which was discovered in the last iteration relays the total packets received to it's undiscovered neighbor nodes after incrementing the path length by *1*, that is it discovers and sends a message to nodes *5* and *6* with path length = *3* and packets = *2*. In the fourth and fifth iterations shown in Figure 4.2(d) nodes *5* and *6* discover node *7 at a* path length = *4* and forward all the packets they received (*2* each) and node *7* combines all the packets to update it's state of having *4* packets at path length =*4*. During the forward propagation phase each

node's forward propagation array is updated with the total number of packets received at that node for certain path length.

Since there are no nodes to discover after the fourth iteration the forward propagation phase stops and the backward propagation phase is initiated, which is shown in Figure 4.3. In the backward propagation phase the nodes are processed in the exact reverse order to that of the forward propagation. Each node that is processed during this phase basically, divides the total number of packets it had received for all path lengths in the proportion amongst the nodes from which it had received packets during the forward propagation phase. Figure 4.3(a) shows the first iteration of this phase where node *7* which was discovered last sends the total packets it had received *(4)* to nodes 5 and 6 in a ratio *2:2* (that is equally) and updates their path length contribution array for path length 4 with total packets = *2.* Similarly in the second iteration of the backward propagation phase shown in Figure 4.3(b) each node *5* and *6* updates the path length contribution array of node *4* for path lengths *3* and *4* by 2 packets from each node for each length. Similarly, Figures 4(c) and 4(d) show iterations 3 and 4 respectively of the backward propagation phase respectively.

α_i^l	i=1	i=2	I=3	i=4	i=5	i=6	i=7
l=1	2/2	2/2	2/2	4/2	2/2	2/2	2/2
l=2	3/3	5/3	5/3	10/3	5/3	5/3	3/3
l=3	4/4	4/4	4/4	8/4	4/4	4/4	4/4
l=4	4/5	2/5	2/5	4/5	2/5	2/5	4/5

Table 4.1: Fractional Contributions (α_i^l) of nodes for the graph in Figure 4.1

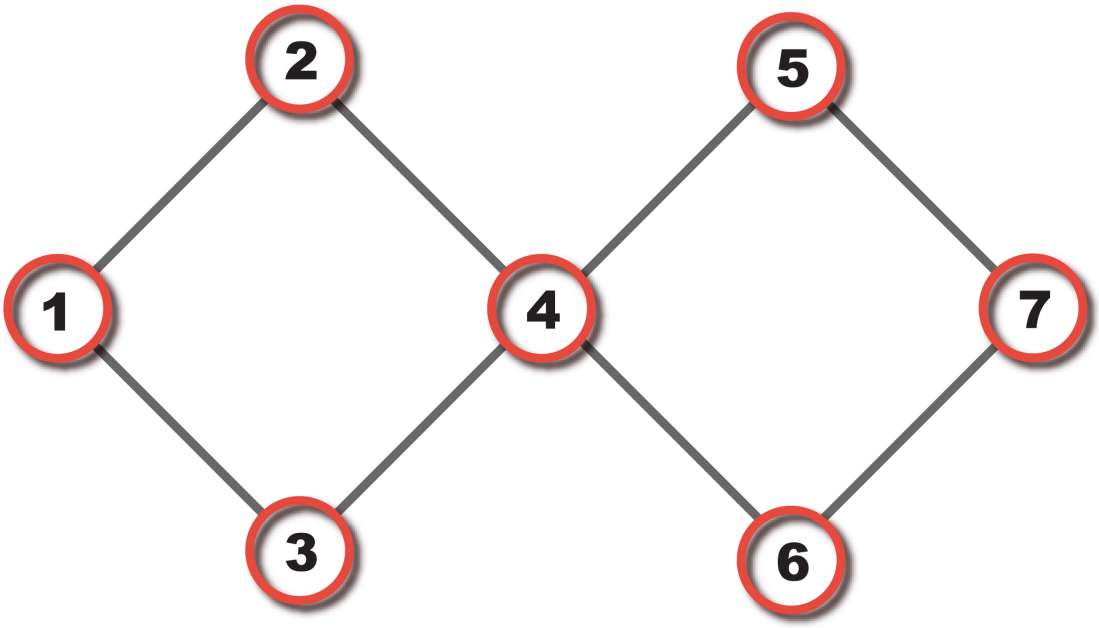


Figure 4.1: Example un-weighted graph for illustration

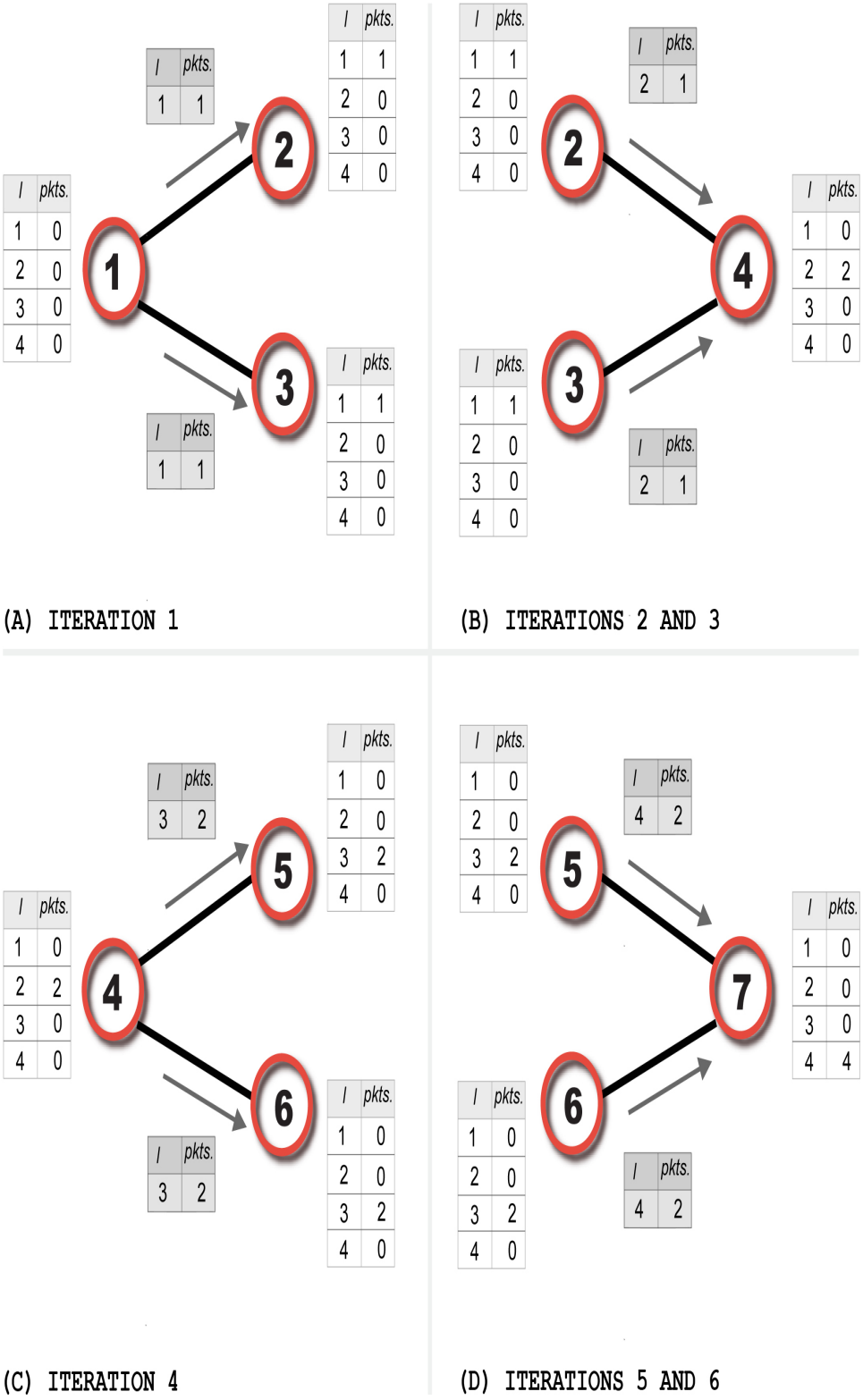


Figure 4.2: Forward Propagation Phase illustration for the example unweighted graph

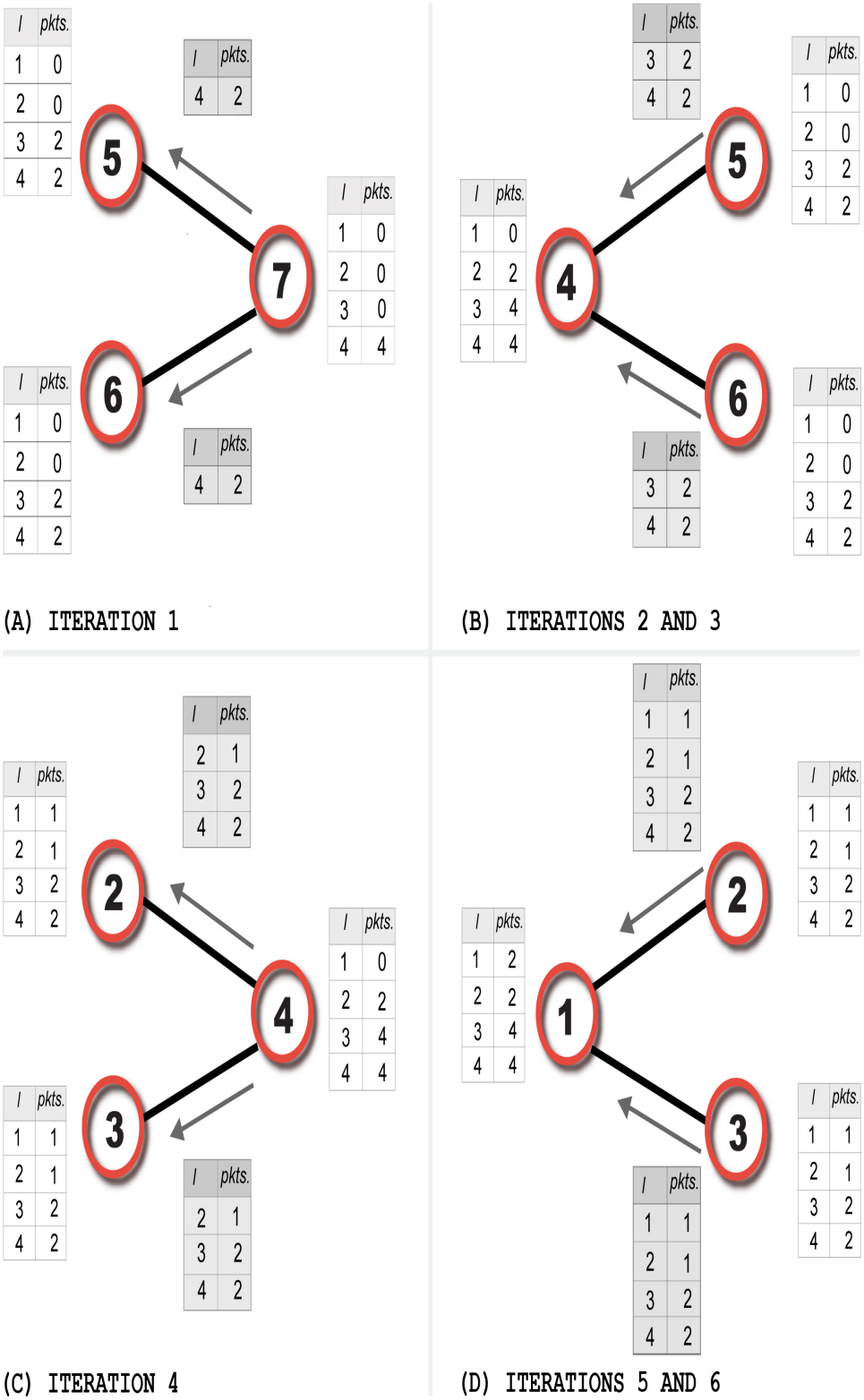


Figure 4.3: Backward Propagation Phase illustration for the example unweighted graph

4.5 Time Complexity Analysis

In this section we discuss in detail the theoretical time complexity of the *NESCap* algorithm given in Algorithm 1. The major assumption made here is that the given graph is sparse, which is a fair assumption made for social graph because of the Dunbar number and preferential attachment phenomenon.

The overall complexity of the algorithm can be determined by adding the complexity of *ComputeFC* (line 2) and the complexity of the nested for loops from line 3 to line 5.

The time complexity of the nested for loops in lines 3 to 5 is basically $O(|V| \times L)$ since the outer loop is for all vertices and the inner loop is for all path lengths to consider till the max path length (which is a user defined parameter and *L <= diameter of graph g*).

The time complexity of *ComputeFC* algorithm listed in Algorithm 2 is basically determined by the for loop in lines *3* to *34* and the for loop from line *35* to *37*. The time complexity of the for loop in lines *3* to *34* is of the order of $O(|V| \times |E| \times L)$ and of the for loop in lines *35*

to *37* is $O(|V|\times L)$. Hence, the complexity of the algorithm *ComputeFC* can be written as following.

$$O(|V|\times |E|\times L + |V|\times L)$$

$$O\left(|V| \times L\left[|E| + 1\right]\right)$$

$$O(|V| \times |E|\times L)$$

Therefore, complexity of the *NESCap* algorithm can be computed as following.

$$O(|V| \times |E|\times L) + O(|V| \times L)$$

$$O(|V| \times |E|\times L)$$

In the worst-case scenario for dense graphs where the number of edges tends towards $|V|^2$ or for graphs where the diameter of the graph tends towards the number of nodes in the graph $|V|$ the worst-case complexity of the algorithm can be of the order $O(|V|^3)$.

But, both these characteristics are not heard of in social networks, as social graphs are sparse and scale free. Hence, for social networks this complexity greatly reduces as the number of edges $|E|$ are of the order of the number of nodes $|V|$ and the diameter is relatively small and can be considered as a constant as is shown by a

phenomenon of six degrees of separation which says that the diameter of a social graph is usually six. Therefore, the time complexity of the algorithms for social graphs would be of the order $\boldsymbol{O}(|\boldsymbol{V}|^2)$.

Chapter 5

Weighted Algorithm

5.1 Introduction

This chapter gives conceptual level details of the algorithm that we have used to implement the valuation-allocation approach for weighted sparse graphs. Social graphs are usually treated as weighted graphs since the strength of ties between individuals can vary a lot and provides crucial information regarding the possibility of information flow along an edge. The philosophy behind the weighted algorithm is on similar lines to that of the un-weighted one. The weighted algorithm also finds all possible shortest paths and distributes the credit of each shortest path amongst the nodes participating in that shortest path. And, it achieves this using two phases; the forward propagation phase identifies the shortest paths and the backward propagation phase divides the value generated by a shortest path amongst the participating nodes. This algorithm can be controlled

with a user provided parameter, which can make the algorithm ignore paths having hops more than the value specified for the parameter. The weighted version of the algorithm is more sophisticated as compared to the un-weighted version. The exact flow of the algorithm is also described using an illustrative example. Finally, an analysis of the computational complexity of the algorithm is provided.

This chapter is organized as follows. Section 5.2 discusses the algorithm in detail. Section 5.3 illustrates this algorithm using an example. To conclude with section 5.4 does an analysis of the theoretical complexity of the algorithm.

5.2 The Algorithm

Our algorithm for computing the social capital value of nodes in a given weighted social graph is listed below in Algorithm 3. The algorithm iterates over all vertices, considering each vertex as the source. It uses Dijkstras [25] approach to find the shortest paths. However, we also need the additional information about which nodes participated in the various shortest paths along with the distance and number of hops in these shortest paths for fair distribution of credit amongst those nodes. The

weighted algorithm also has two phases namely the forward propagation phase (line *3* to line *29*) and the backward propagation phase (line *30* to *37)*.

Though, logically it is pretty similar to the algorithm for un-weighted graphs but has some key differences. Those differences are as following. The first is that a Fibonacci heap priority queue is used during the forward propagation phase of the weighted algorithm as compared to a simple FIFO queue that was used in the un-weighted algorithm. This is because two nodes can have a shorter path, which has more number of hops as compared to another path, which is longer (more distance) but has less number of hops. The second is that now a node can be a part of multiple shortest paths in between two specified nodes, where the distance of the paths would be the same but the number of hops can be different, and hence this information would also need to be maintained in the state of the node. For the un-weighted case since the distance and hops were the same hence no such issue was to be taken care of.

5.2.1 Forward Propagation Phase

The forward propagation phase is essentially on the same lines as for un-weighted graphs (described in section

4.2.1). During this phase will essentially find all the shortest paths between the current iteration source vertex and all other vertices connected to it within the maximum number of hops specified by the user. Here we use a Fibonacci heap priority queue to manage the vertices, which are still to be expanded (priority used in this queue is the distance from the source vertex). Each vertex on expansion passes the messages (also called packets) it has received to its neighboring vertices incrementing the hops of the message by one and the distance by the inverse of the edge weight. Each vertex accumulates these messages in a map (or an array) using hops as the key and maintains the count of the number of packets for a given hop value as the value.

5.2.2 Backward Propagation Phase

The backward propagation phase relays information back from the terminal nodes of the shortest paths in the reverse chronological order of discovery. Every vertex picked from the stack sends messages back to the immediately preceding vertices from which it received packets. In the case of backward propagation these packets are accumulated using distance and hops as the key. The distance is the shortest distance from the source vertex to the terminal vertex where the packet was

initiated. The hops are counted from the backward propagation vertex to the current vertex.

When a vertex is selected from the stack for backward propagation the forward and backward messages for the vertex will together have complete information about all the shortest paths on which this vertex appears for a given source vertex. The cross product of the forward message and backward message set for the selected vertex can be *used* to compute the social capital value contributed by all shortest paths passing through this vertex. This process is shown in Algorithm 4 below.

Algorithm 3: SoCap

Input: G(V,E): social nettwork graph; L: maximum path length

Output ; $Y=[y_1,\ldots,y_m]$: social capital of nodes in V

```
1  for each v in V do
2      Initialize Queue Q, Stack S;
3      v.status ⟵ found; v.dist⟵ 0; v.fpmsgs ⟵ [ ]; v.bpmsgs ⟵[ ];
       v.activeICEdges ⟵ [ ];
4      v.fpmsgs.add(hops ⟵ 0,pkts ⟵ 1;
5      Q.add(v);
6      While ( !Q.empty() ) do
7          u ⟵ Q.poll();
8          u.status⟵ closed;
9          S.push(u);
10         if u.fpmsgs.minHops() = = L then
11             continue;

12         for each w adjacent to u do
13             e = E[ u, w]; /*E[u,w] is edge from u to w*/
14             if w:status = = closed then
15                 continue;

16             if w.status = = found then
17                 if w.dist < u.dist + (1/e.weight) then
18                     continue;

19                 if w.dist > u.dist + (1/e.weight) then
20                     w.fpmsgs ⟵ [ ];
21                     w.activeICEdges ⟵[ ];
22                     w.dist ⟵ u.dist + (1/e.weight);
23                     Q.decrease Priority (w);

24             else
25                 w.dist ⟵ u.dist + (1/e.weight); w.status ⟵ found; Q.add(w);

26             for each msg in u.fpmsgs do
27                 if msg.hops() = = L then
28                     continue;

29                 w.fpmsgs.add( hops ⟵ msg.hops() + l. pkts ⟵ msg.pkts );
                   w.activeICEdges.add(e);

30     while (!S.empty()) do
31         u ⟵ S.pop();
32         u.bpmsgs.add( dist ⟵ u.dist, hops ⟵ 0,pkts ⟵ 1);
33         u.scv ⟵ u.scv + computeSCV(u);
34         for each e( w , u) in u.activeICEdges do
35             for each msg in u.bpmsgs do
36                 w.bpmsgs.add( dist ⟵msg.dist, hops ⟵msg.hops + 1, pkts ⟵
                   msg.pkts);

37         u.status⟵ not found; u.dist⟵ ∞ ; u.fpmsgs ⟵ [ ]; u.bpmsgs⟵ [ ];
           u.activeICEdges⟵ [ ];

38 for each v in V do
39     Y.add(v.scv);

40 Return Y;
```

Algorithm 3: Pseudo code for SoCap Algorithm for weighted sparse graphs

Algorithm 4: computeSCV

Input: v: vertex to compute Social Capital For
Output: $iterSCV$: social capital of vertex in V for the current source vertex

```
1 currentSCV ← 0;
2 for each fpmsg in v.fpmsgs do
3     for each bpmsg in v.bpmsgs do
4         dist ← bpmessage.dist;
5         totalPkts ← fpmsg.pkts * bpmsg.pkts;
6         totalHops ← fpmsg.hops + bpmsg.hops;
7         currentSCV ← currentSCV + (totalPkts * b(dist) / (totalHops + 1)) ;
8 Return currentSCV;
```

Algorithm 4: pseudo code of the algorithm for computing the social capital value at a node in a weighted sparse graph by multiplying forward propagation messages by backward propagation messages

5.3 Illustrative Example

Let us illustrate the working of the algorithm using the weighted graph given in Figure 5.1. We will take a look at the run of the algorithm, keeping vertex *1* as the source vertex. First of all the forward propagation phase of the algorithm is invoked, which basically searches for shortest paths originating from the source node *1*. The first step of the forward propagation phase adds a dummy forward-message to the source node *1* the contents of the message are *{distance = 0, hops = 0, packets =1}*, and then this node is inserted in the priority queue. In the second step source node *1* is polled from the front of the priority queue and it sends forward propagation messages to each of its non-closed neighbors *2, 3* and *4* as shown

in Figure 5.2 (a). Now since nodes *2* and *3* are at a higher priority than node 4, each of them is expanded and they relay the forward propagation packets to node *4*, which already exists in the priority queue and is updated. This is shown in Figure 5.2 (b). We can observe that node *4* exists at a shortest distance of 4 from the source node *1* once via a path of a single hop *{1,4}* and twice through paths of two hops *{1,2,4}* and *{1,3,4}*. Finally, node 4 is expanded from the priority queue and sends a forward propagation message to node *5* this message contains two components one comprising of reaching node *5* in 2 hops and the other of reaching node *5* in 3 hops. This is shown in 5.2 (c). Since, there are no more non-closed nodes left therefore the queue becomes empty and the forward propagation phase comes to an end. All the shortest paths with node 1 as the source node have been determined at this point.

Now, the backward propagation phase of the algorithm gets invoked. In the backward propagation phase the nodes are processed in the reverse order to that of the forward propagation phase. And, this works differently from the backward propagation phase of the un-weighted algorithm. Here, the current terminal node simply adds a dummy backward message to itself, the contents of the dummy message are *{distance = discovery-distance, hops = 0,*

packets = 1}. And the entire backward propagation message list (or array) is sent back along all the active incoming edges (edges which were a part of the shortest paths which were discovered in the forward propagation phase). Then the computeSCV method is called for the terminal node, which updates the social capital value for the terminal node (for the current source node) by taking a *product* of the forward propagation messages and backward propagation messages present on that node. Now, the temporary fields of this terminal node's state are reset. Figure 5.3 (a) shows this occurring for the terminal node *5*, which adds a dummy backward propagation message to itself *{distance = 5, hops = 0, packets = 1}* and sends node *4* a backward propagation message by incrementing hops by 1. After this social capital value for 5 is computed and it's temporary fields are reset. Then in the next step shown in Figure 5.3(b), the backward propagation algorithm starts with node *4* as the terminal node. This node then adds a dummy backward propagation message to itself *{distance = 4, hops = 0, packets = 1}* and has two backward propagation messages now. Then, the computeSCV method is called for node *4* and this node's temporary fields are cleared. Similarly, further steps, which are the logical extension to the above steps, happen for nodes *2* and *3* shown in Figure 5.3(c). With this the backward propagation phase

concludes with each node having the social capital value due to all shortest paths originating from the source vertex. Once, we iterate over all vertices as the source we get the final allocated social capital value for each node due to the entire graph.

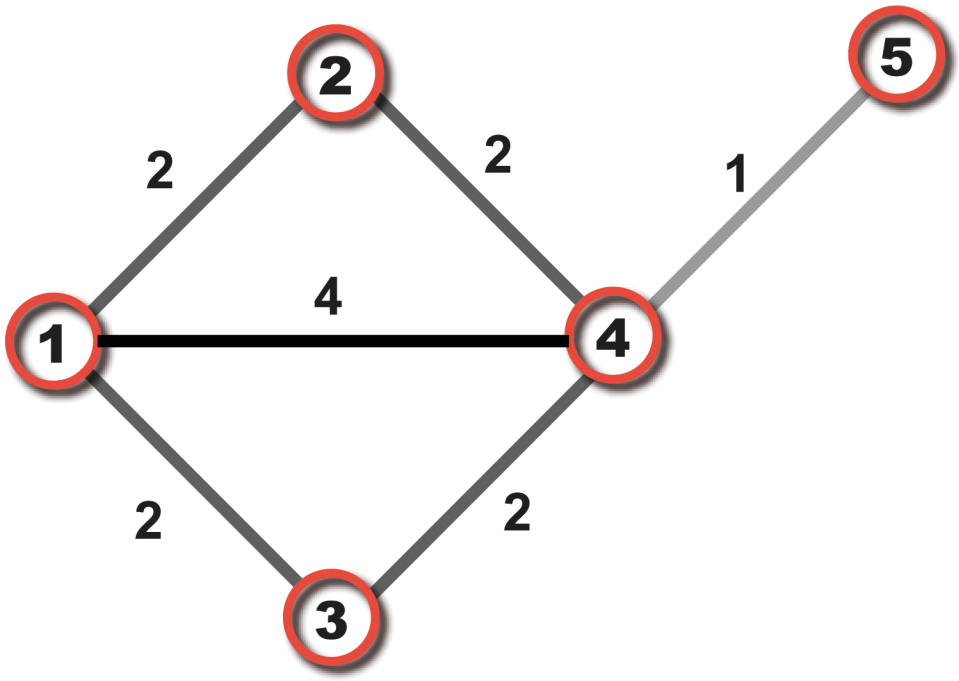
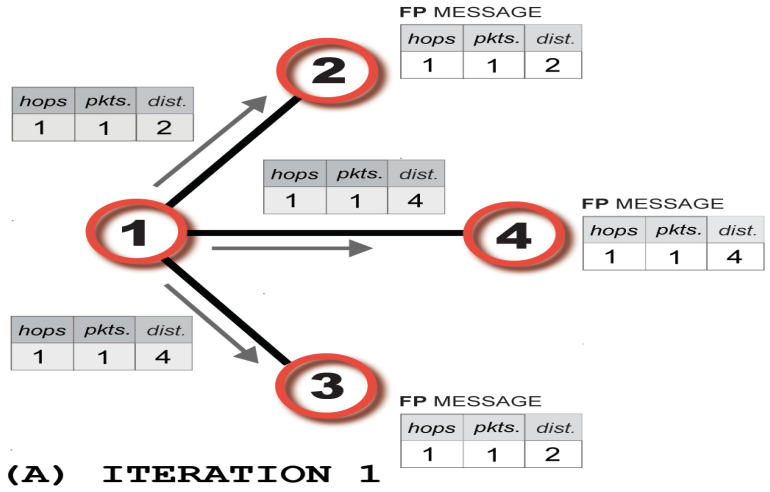
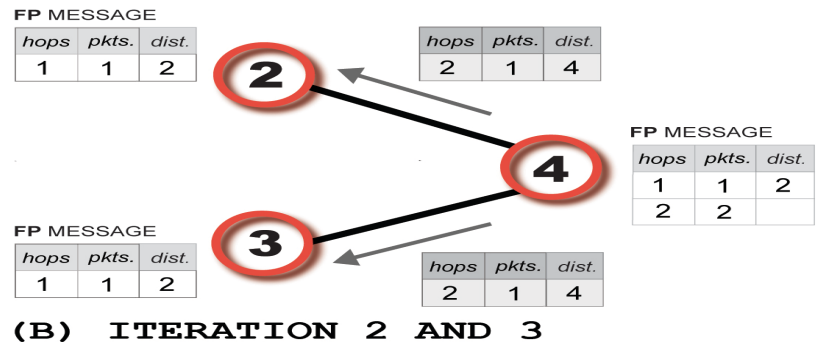


Figure 5.1: Example Weighted Graph for Illustration





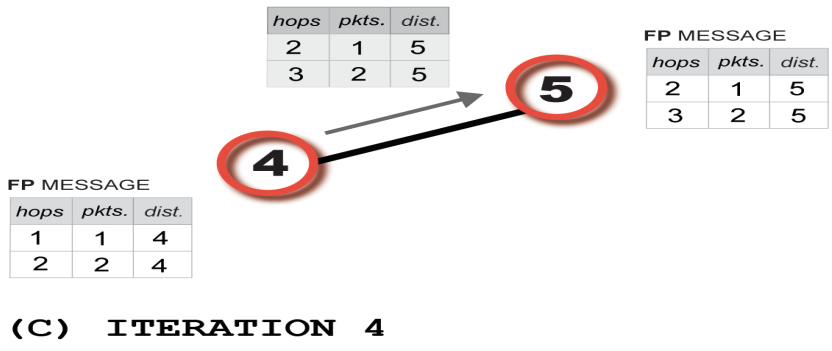
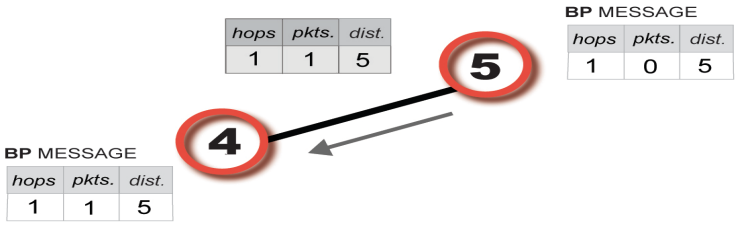
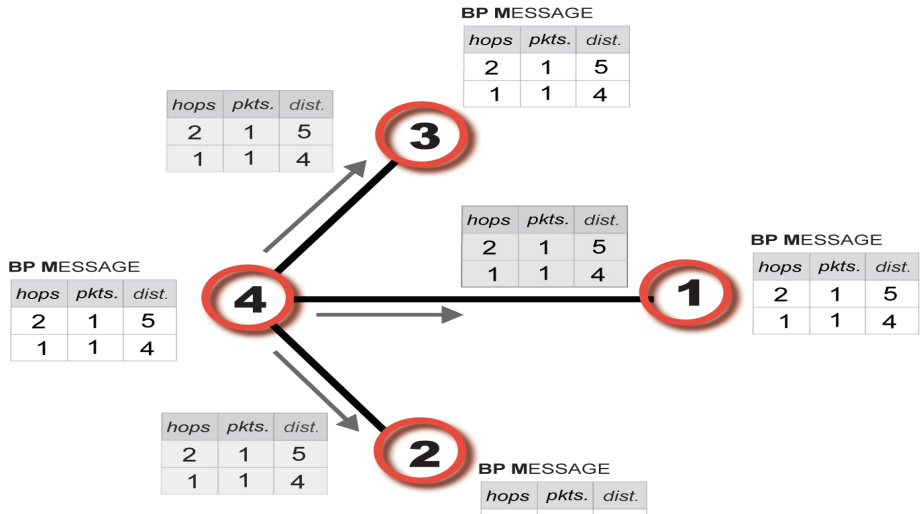


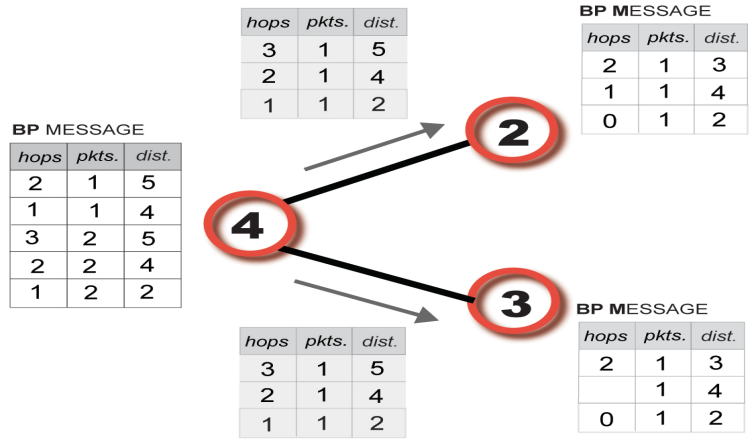
Figure 5.2: Forward Propagation Phase Illustration for the example Weighted Graph



(A) ITERATION 1



(B) ITERATION 2



(C) ITERATION 3 AND 4

Figure 5.3: Backward Propagation Phase Illustration for the example Weighted Graph

5.4 Time Complexity Analysis

In this section we discuss in detail the theoretical time complexity of the *SoCap* algorithm. The complexity of the forward propagation phase is due to the discovery of the shortest paths from a given source vertex, which is repeated considering each vertex as a source. This can be seen in the *SoCap* algorithm from line *1* to line *29*. This complexity can be written as below.

$$O\left(|V| \times \left(|V| \times \log|V| + |E| \times \log|V| + |E| \times \frac{|E|}{|V|}\right)\right)$$

Since we are talking about sparse scale free networks we can assume that the number of edges $|E|$ is of the order of $|V|$ itself. Hence, the above equation can be written aa following.

$$O\left(|V|^2 \times \log|V|\right)$$

For the backward propagation phase listed in Algorithm 3 from line 30 to line 37. This includes a call to *computeSCV* listed in Algorithm 4. The complexity can be written as follows.

$$O\left(|V| \times [|V| \times L \frac{|E|}{|V|} \times |V| + |E| x |V| X L]\right)$$

Which is dominated by the cubic term. Hence, the complexity of the backward propagation can be simplified to the following.

$$O(|V|^3)$$

Since, this dominates over the complexity of the forward propagation phase as well, we can conclude that the worst case complexity of the algorithm is $O(|V|^3)$.

Chapter 6

Weighing Schemes

6.1 Introduction

This chapter discusses the various possible weighing schemes, which can be used to measure the strength of ties between individuals (nodes) in a social network (graph). It is vital to capture these tie strengths since not all ties are equal in strength and the stronger a tie is the easier is the path it offers for information to flow through that edge. For example, in a collaborative network if two authors frequently publish papers together they are more likely to have a stronger tie as compared to authors who have only published one paper together.

Furthermore, numerous real-world social interactions involve multiple people. For example, authors collaborating on a paper, teams playing together in multiplayer games etc. Using simple graphs to model these activities does not accurately capture their inherent

group structure. Hypergraphs have recently emerged as a better tool for modeling such interactions. In this chapter we discuss various schemes to measure the weights of hyperedges and compare them empirically. Finally, we give an algorithm to collapse a given hypergraph into a simple (yet more informative) graph structure.

This chapter is organized as follows. Section 6.2 talks about the simple graph based weighing scheme, Section 6.3 gives an overview of hypergraph basics, Section 6.4 discusses the conversion process of a hypergraph to a graph with additional information, and Section 6.5 discusses in detail the various hypergraph weighing schemes we have studied. The Chapter concludes with Section 6.6, which focuses on developing a process for selecting the best weighing scheme.

6.2 Simple Graph Weighing Scheme

This section talks about the approach we have taken to measure the strength of ties using simple dyadic graph structure as the modeling tool. Though it’s less powerful as compared to a Hypergraph, it has still proved to be a powerful tool for doing various kinds of analysis on social networks and various other domains.

In our model we simply use the number of interactions between two people (nodes) as the strength of tie between them even though these interactions might have involved multiple actors. For instance, in a co-authorship network if two authors have co-authored *n* papers then the weight of that edge would simply be *n*.

Let us take an example to illustrate this concept. Let us assume we have four authors in our network A, B, C and D. They have written four papers P1, P2, P3 and P4 together. This scenario is listed in Table 6.1.

Paper	Collaborating Authors
P1	A, B, C
P2	A, B
P3	B, D
P4	A, B, C
P5	A, B, D

Table 6.1: Example listing of papers for a co-authorship network to illustrate simple graph-weighing scheme

Using the papers listed in table 6.1 the resulting weighted graph is shown in Figure 6.1. It can be seen that each edge has the weight that is equal to the number of papers (interactions) written together.

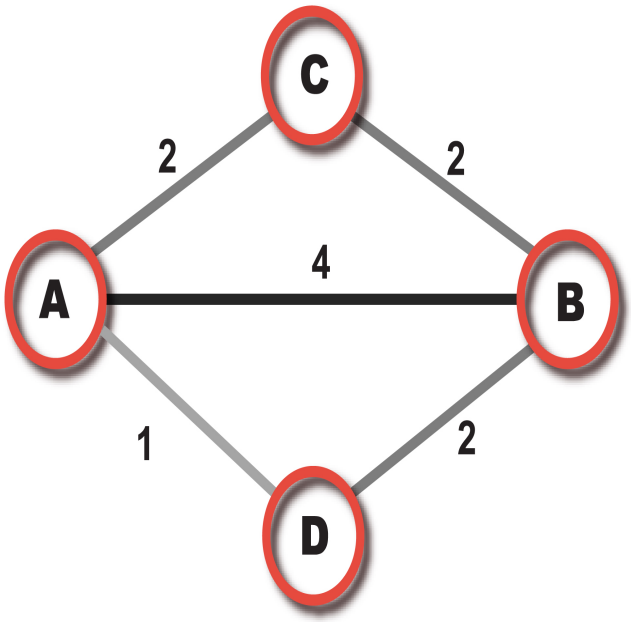


Figure 6.1: Example weighted graph representation of the sample co-authorship network listed in Table 6.1

6.3 Hypergraphs Overview

Hypergraphs [26] are a generalization of simple graphs, which have been used in various domains to model higher order (n-adic) relationships [27] [28] [29].

Mathematically, a Hypergraph can be represented as follows.

$$H = (V, HE)$$

$$h_i \subseteq V, \forall\, h_i \in HE$$

Where, V is the set of Vertices and HE is the set of Hyperedges. Each Hyperedge is itself a subset of the set of vertices (V), which provides a natural representation

for higher order relationships. Hypergraphs are more natural and complete when it comes to modeling such group level interactions, which are commonplace in real-life social networks.

Figure 6.2 depicts the Hypergraph representation of the co-authorship network listed in Table 6.1.

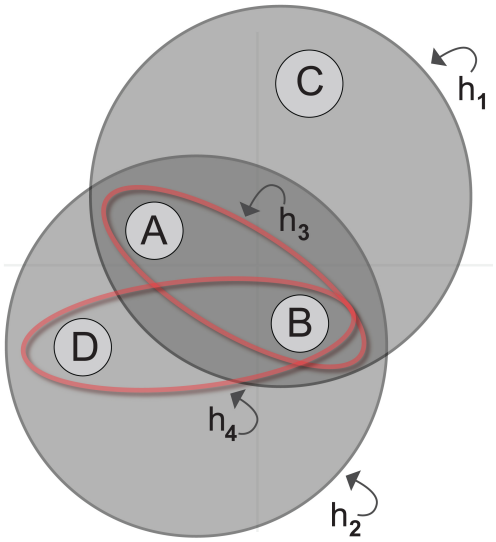


Figure 6.2: Example hypergraph representation of the sample co-authorship network listed in Table 6.1

6.4. Converting a Hypergraph to a Graph

Hypergraphs are great for modeling group level phenomenon where multiple actors interact and hence higher order relationships are brought into the picture. But after modeling these relationships using hypergraphs we can reduce the dimensionality of these hypergraphs to utilize

graph algorithms on top of these condensed graphs. [31] Talks about a shortest path algorithm using the concept of dimensionality reduction with some extra information, which is captured using hypergraphs as a modeling tool. In this section we introduce an algorithm that does this conversion while keeping the extra information intact.

Mathematically, the weight of an edge *e(I,j)* in the condensed graph (or dimensionality reduced graph) can be defined as follows.

$$w(i,j) = \sum_{i \in h, j \in h, h \in HE} w_h$$

This equation specifies that any edge *e(i,j)* in the condensed graph will have a weight equal to the sum of the weights of all hyperedges of which the nodes *i* and *j* are a part of. The algorithm for this conversion is provided below.

Algorithm 5: HypergraphToGraphConverter

Input: $H(V, HE, W)$: weighted hypergraph
Output: $g = G(V, E, W*)$: condensed graph of H
Initialize weighted graph g;
for $v_i \in V$ **do**
  for $v_j \in V$ *and* $v_i \neq v_j$ **do**
    for $h \in H$ **do**
      if $v_i \in h$ *and* $v_j \in h$ **then**
        $e \leftarrow$ new edge (i, j);
        if $g.containsEdge(e)$ **then**
          $wt \leftarrow g.getEdgeWeight(e) + w_h$;
          $g.updateEdgeWeight(e, wt)$;
        else
          $wt \leftarrow w_h$;
          $g.addEdge(e, wt)$;
Return g;

Algorithm 5: Algorithm for converting a weighted hypergraph into a condensed weighted graph

6.5 Hyperedge Weighing Schemes

Hyperedge weights are the measure of the strength of the tie amongst the members of the hyperedge due to that particular hyperedge. There are many possible ways of measuring the weight of a Hyperedge. Generally, the weight of a Hyperedge is dependent upon two factors, namely, multiplicity and cardinality of the hyperedge.

Multiplicity of a Hyperedge h_i is denoted as m_i and is the total number of times a particular set of people (nodes) have interacted or the frequency of the occurrence of the hyperedge. For example the Hypergraph represented in Figure 6.2 (from the listing in Table 6.1) we can see

that there are four unique hyperedges, which are listed in Table 6.2, and hyperedge h_1 is repeated twice (due to papers P1 and P4). Hence the multiplicity (m_1) of h_1 is 2.

Cardinality of a hyperedge h_i is denoted as c_i and is the total number of nodes that are members of that hyperedge. In set notation it can be written as $c_i = |h_i|$. The multiplicity and cardinality values for the Hypergraph shown in Figure 6.2 are listed in Table 6.2 below. Any hyperedge that has a cardinality of 2 is a simple dyadic edge. It is worth nothing here that a hyperedge can also have a cardinality of 1, for example in case when an author writes a paper alone.

Hyperedge	Authors	Multiplicity	Cardinality
H_1	A, B, C	2	3
H_2	A, B, D	1	3
H_3	A, B	1	2
H_4	B, D	1	2

Table 6.2: Listing of the cardinality and multiplicity of various hyperedges of the hypergraph in Figure 6.2

The following sections list the various possible schemes, which can be used for weighing hyperedges in a hypergraph.

6.5.1 Constant

This scheme is analogous to the un-weighted cases for graphs. We simply assign each Hyperedge a constant unit weight, w_j = 1. This scheme basically states that all the hyperedges are of equal unit strength. Table 6.3 lists the weights for the Hypergraph in Figure 6.2 using the constant scheme.

Hyperedge	Authors	Multiplicity	Cardinality	Weight
H_1	A, B, C	2	3	1
H_2	A, B, D	1	3	1
H_3	A, B	1	2	1
H_4	B, D	1	2	1

Table 6.3: Listing of weights using the constant scheme for various hyperedges of the hypergraph in Figure 6.2

6.5.2 Frequency Based

In this weighing scheme each hyperedge is assigned a weight equal to the multiplicity of that hyperedge. This scheme can be thought as being analogous to the simple graph-weighing scheme where a tie (edge) has the strength equal to the number of interactions that have happened between the people (nodes) connected by that tie. Mathematically it can be formulated as following.

$$w_j = m_j$$

Table 6.4 lists the weights for the Hypergraph in Figure 6.2 using this frequency-based scheme.

Hyperedge	Authors	Multiplicity	Cardinality	Weight
H_1	A, B, C	2	3	2
H_2	A, B, D	1	3	1
H_3	A, B	1	2	1
H_4	B, D	1	2	1

Table 6.4: Listing of weights using the frequency-based scheme for various hyperedges of the hypergraph in Figure 6.2

6.5.3 Newman's Definition

Newman [30] defined a weighing measure, which measured the strength of ties between authors (nodes) in a collaboration network. A collaboration network is essentially a bi-partite network where one set of nodes represents authors (or collaborating entities) and the other set of nodes represents collaborations. He defined the strength of a tie between two authors (nodes) who have collaborated on a paper as following.

$$w_{ij} = \sum_k \frac{\delta_i^k \delta_j^k}{|c_k| - 1}$$

Where i and j are the collaborating authors. The symbol w_{ij} represents the strength of the tie between authors i and j. δ_p^q represents if author (node) p is a member of collaboration q. k represents all the collaborations in the given collaboration network and $|c_k|$ represents the number of authors (or the cardinality) in a given collaboration k.

We have ported Newman's definition from bi-partite collaborative networks to Hypergraphs. In our formulation we say that the weight (w_j) of a hyperedge (h_i) is directly proportional to the multiplicity (m_i) of the hyperedge and inversely proportional to the cardinality (c_i) of the hyperedge. This definition is consistent with Newman's definition. Mathematically it can be written as following.

$$w_i = \frac{m_i}{|c_i| - 1}$$

Table 6.5 lists the weights for the Hypergraph in Figure 6.2 using Newman's collaborative networks based scheme.

Hyperedge	Authors	Multiplicity	Cardinality	Weight
H_1	A, B, C	2	3	1
H_2	A, B, D	1	3	0.5
H_3	A, B	1	2	1
H_4	B, D	1	2	1

Table 6.5: Listing of weights using the Newman's Collaborative networks based scheme for various hyperedges of the hypergraph in Figure 6.2

6.5.4 Gao's Definition

Gao [35] defined hyperedge weights using the Enron e-mail dataset. In their formulation the weight of a hyperedge was a measure of the distance (inverse of weight) between the nodes that were affiliated with that hyperedge. For them a Hyperedge was nothing but an e-mail thread and the nodes were nothing but people on that e-mail thread. We take an inverse of their weight definition to make it consistent with our interpretation of hyperedge weights, which represent the strength of ties between the nodes in a hyperedge. Our formulation can be mathematically be represented as the following.

$$w_i = \left(\sqrt{|c_i|}\right)^{\beta^{m_i-1}}$$

Where, β is a constant.

Table 6.6 lists the weights for the Hypergraph in Figure 6.2 using Gao's Enron hyperedge based scheme using $\beta = 2$

Hyperedge	Authors	Multiplicity	Cardinality	Weight
H_1	A, B, C	2	3	3
H_2	A, B, D	1	3	1.73
H_3	A, B	1	2	1
H_4	B, D	1	2	1

Table 6.6: Listing of weights using the Gao's Enron hyperedge based scheme for various hyperedges of the hypergraph in Figure 6.2 using $\beta = 2$

6.5.5 Network Theory Definition

We can define weights of a hyperedge probabilistically using network theory. Hyperedge weights can be defined as the probability of contact between any two nodes over m (multiplicity) interactions. The probability of the event of contact between any two nodes over m trails is equivalent to the probability of event that no contact is formed between those nodes over m trials. Mathematically this can be formulated as the following.

$$w_i = 1 - \left(1 - \frac{1}{c_i - 1}\right)^{m_i}$$

Table 6.7 lists the weights for the Hypergraph in Figure 6.2 using network theory based scheme.

Hyperedge	Authors	Multiplicity	Cardinality	Weight
H_1	A, B, C	2	3	3
H_2	A, B, D	1	3	1.73
H_3	A, B	1	2	1
H_4	B, D	1	2	1

Table 6.7: Listing of weights using network theory based scheme for various hyperedges of the hypergraph in Figure 6.2

6.6 Selecting the Appropriate Weighing Scheme

This chapter works towards selecting the most appropriate weighing scheme from amongst the five weighing schemes described in the previous section. We have developed a methodology for the same and have conducted relevant experiments as well.

6.6.1 Overall Process

In this subsection we describe the overall methodology we have developed for selection of the most appropriate weighing measure for our scenario. The process is described below.

The first step in this process is to weigh the given hypergraph *H* using the different weighing schemes. Then, once we obtain a weighted hypergraph we convert this into a simplified weighted graph representation using the Algorithm (Algorithm 6.1) in section 6.4. Using this weighted graph definition we compute the degree centrality for each node and co-relate this degree score with the given influence ground truth score. We have used two datasets; one real world DBLP collaboration network and the other a virtual world network that is the CR3 questing network to determine the best weighing measure. Both these datasets contain multi actor relationships and are ideally suited to be modeled using hypergraphs. Our definition of 'the most appropriate' is based on the assumption that a centrality measure is a direct measure of some sort of influence and the best weighing measure should have the highest correlation (Pearson correlation co-efficient) with the influence ground truth (given in the dataset). Figure 6.3 shows these steps as in a simple process flow diagram.

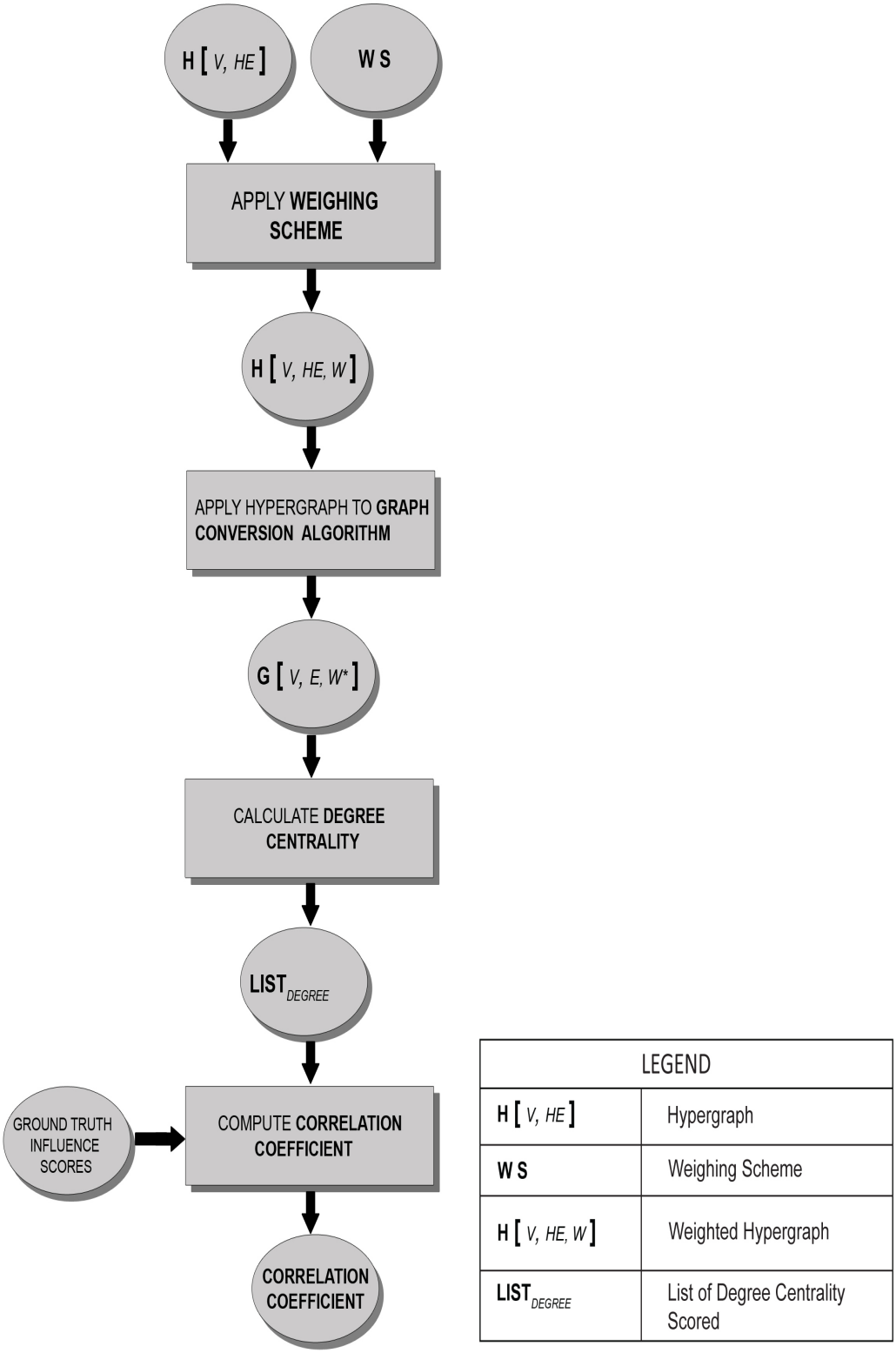


Figure 6.3: Process flow diagram representing the overall process for determining the optimal hyperedge weighing scheme

6.6.2 Dataset Description

We have used two datasets one from the real world that is the DBLP co-authorship dataset and one from the virtual worlds that is the CR3 group network.

DBLP Dataset: DBLP is a computer science bibliography dataset, which has data about various publications in the field of computer science. The nodes in this network represent authors and the hyperedges connect the authors who have connected over a publication. We have used a three-year snapshot of the data for creating the hypergraph. This hypergraph contains 174,367 nodes (authors) and 175,345 hyperedges (collaborations) as listed in Table 6.8. We use the total count of citations obtained by an author in the given time period as our target variable for verification (or as the ground truth of influence). The citation counts received by an author is obtained by summing over the citations received by the publications the author is present in (this is provided in the dataset).

CR3 Dataset: CR3 is a very popular Chinese Massively Multiplayer Online Role Playing Game (MMORPG). We use the group network present in CR3 as a hypergraph, where a

node represents a player in CR3 and a hyperedge connects the members of a group formed by players for questing. We use data of one-week snapshot of the group data for creating the hypergraph. The resulting hypergraph contains 29,438 nodes (players) and 233,092 hyperedges (groups) as listed in Table 6.8. We also have the friendship request for the CR3 data for this period and we use the total number of friendship requests received by a player (node) as the target variable for verification (or the ground truth for influence).

Dataset Name	Period of dataset	Number of Nodes	Number of Hyperedges
DBLP	3 years	174,367	175,345
CR3	1 week	29,438	233,092

Table 6.8: Statistics for DBLP and CR3 datasets used for finding the best weighing measure by using correlation with ground truth of influence

6.6.3 Experiments and Results

Using the dataset described in the section above we do a simple correlation analysis between the degree centrality obtained between the different weighing measures and the influence ground truth score of these nodes. The Pearson correlation coefficient scores for the DBLP dataset are listed in Table 6.9 and the Pearson correlation

coefficient scores for the CR3 dataset are listed in Table 6.10.

For both the datasets it is observed that when Newman's weight definition scheme is used, the resulting degree centrality has the highest correlation with the influence ground truth numbers. Hence, we can conclude that Newman's weighing definition is the most suitable.

Hyperedge Weighing Scheme	Correlation Coefficient
Constant	0.391
Frequency-based	0.413
Newman's	**0.441**
Gao's ($\boldsymbol{\beta = 0.3}$)	0.427
Network	0.403

Table 6.9: Hyperedge weighing schemes and the correlation score of degree centrality using these schemes and ground truth influence score for the DBLP dataset

Hyperedge Weighing Scheme	Correlation Coefficient
Constant	0.575
Frequency-based	0.579
Newman's	**0.593**
Gao's ($\boldsymbol{\beta} = \mathbf{1.0}$)	0.590
Network	0.571

Table 6.10: Hyperedge weighing schemes and the correlation score of degree centrality using these schemes and ground truth influence score for the CR3 dataset

Chapter 7

Results

7.1 Introduction

This Chapter elaborates upon the various experiments that have been performed for testing the effectiveness and efficiency of the method we have proposed as compared to several other popular baseline methods. We have used two real world collaboration networks to evaluate our method, US Patent collaboration network dataset and DBLP co-authorship network dataset. We have used PMIA, Weighted Degree and PageRank as the baseline methods, which we compare against our method. The evaluation criterions are Precision at K, Recall at K and F-Measure at K, which, are standard criterions, used in Information Retrieval literature. Our method performs quite well as compared to the baselines, especially in the DBLP dataset where we beat the baselines by a margin of more than 8%. We also conduct these experiments using weighted hypergraph modeling and observe that this modeling gives slight

improvement in the evaluation criterions as compared to the corresponding weighted graph alternatives. We also performed a small case study of the top 10 authors yielded by the various methods. Furthermore, we do a comparison of the run times of our method against the other baselines.

This chapter is organized as follows. Section 7.2 describes the datasets that we have used. Section 7.3 describes the various evaluation measures, which we have used. Section 7.4 does the effectiveness analysis of our algorithms compared to the other baselines. Section 7.5 presents a small case study. To conclude with section 7.6 provides an efficiency analysis of our algorithm as compared to the baselines.

7.2 Datasets Used

We have used two collaborative networks namely the DBLP dataset and US Patent dataset to compare and contrast the various aspects of performance of our method to the other baselines. This section describes both these datasets in detail.

7.2.1 DBLP Dataset

DBLP is a computer science bibliography dataset. It contains information about which authors have collaborated together to publish in the field of computer science and the citations received by each of such publications.

The downloaded DBLP dataset [32] contains information about all computer science publications over a 75-year period starting from 1936 up to 2011. The dataset has 1,033,321 distinct authors and 1,632,443 publications. We constructed the corresponding DBLP weighted graph (using both simple graph modeling and hypergraph modeling) where nodes represented authors and edges represented the tie strength between these authors (due to the papers they have published together). This graph had 1,033,321 nodes and 3,489,607 undirected edges. Out of these 58,277 nodes did not have any edges, which means that these authors only, wrote papers solo, without any co-authors. On an average each author was connected to 3.38 other authors (also called the average degree) and the entire graph had 104,299 weakly connected components, including the isolated nodes (nodes with zero degree). The degree distribution of DBLP data is shown in Figure 7.1(a) and

the power law exponent of this degree distribution was 2.4672. Details about the DBLP data are also listed in Table 7.1.

7.2.2 US Patent Dataset

The other dataset used for our analysis is the US Patent Dataset. This dataset contains information about which authors have collaborated together to write US Patents. This dataset also contains information regarding the citations received by each patent.

The downloaded US Patent dataset [33] contains information about all patents granted from 1977 to 1999 by the US Patent Office (USPTO). The patent co-authorship network has 1,357,542 distinct authors and 1,864,794 patent publications. We constructed the corresponding US Patent weighted graph (using both simple graph modeling and hypergraph modeling) where nodes represented authors and edges represented the tie strength between these authors (due to the patents they have published together). This graph had 1,357,542 nodes and 2,509,120 undirected edges. Out of these 291,660 nodes did not have any edges, which means that these authors only, published patents solo, without any co-authors. On an average each

author was connected to 1.85 other authors (also called the average degree) and the entire graph had 417,933 weakly connected components, including the isolated nodes (nodes with zero degree). It is noteworthy here that the US Patent co-authorship graph is much sparser as compared to the DBLP co-authorship graph, the average degree of US Patent graph is around 1.8X lower as compared to the DBLP graph. The degree distribution of US Patent data is shown in Figure 7.1(a) and the power law exponent of this degree distribution was 2.4672. Details about the US Patent data are also listed in Table 7.1.

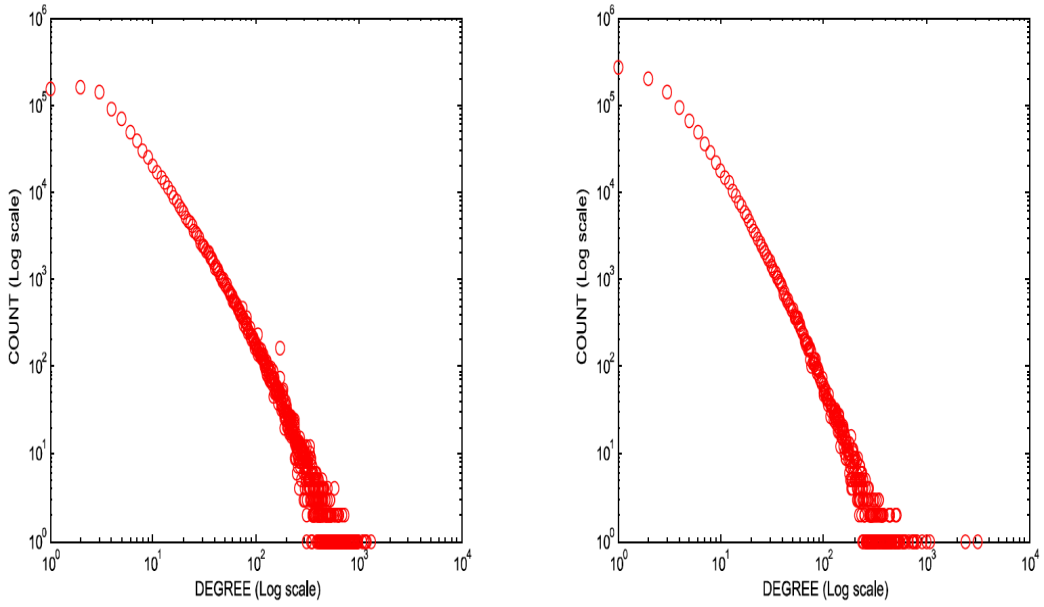


Figure 7.1: Degree Distribution of DBLP and US Patent graphs

Dataset	DBLP	US Patent
# Nodes	1,033,321	1,357,542
# Publications	1,632,443	1,864,794
# Edges	3,489,607	2,509,120
# Isolated Nodes	58,277	291,660
# Weakly CC	104,299	417,933
Avg. Degree	3.38	1.85
Min Pub. Date	1936	1977
Max Pub. Date	2011	1999
Period (years)	75	22

Table 7.1: Statistics for DBLP and US Patent Data

7.3 Baselines

We have compared our method against three baselines, each having different characteristics. These algorithms are popularly used as baselines for comparison in influence maximization problems [4][5]. These three choices are justified by the variety they offer. PMIA is a more recent influence cascade based algorithm, PageRank is a popular variant of Eigenvalue Centrality and Weighted Degree is based on the degree of a node. Each of these baselines is briefly described below.

***PMIA*:** It is the prefix-excluded extension of Maximum Influence Absorbance (MIA) model [5]. This algorithm takes a network structure with pre-defined edge probabilities, which basically represent the probability of influence flow along an edge. We have used the weighted cascade model proposed in [4] to compute these edge probabilities.

Weighted Degree: This is a very straightforward baseline, which says that influencing capability of a node is directly proportional to the weighted out degree it has. Hence, the top-k influential nodes selected by this method are the top-k nodes with highest weighted out degree.

PageRank: PageRank is an immensely popular variant of Eigenvalue Centrality. We have used the power method to compute the page rank values of nodes. In our experiments the restart probability was set to *0.15* and the stopping criteria that is the distance between L1 norm between two consecutive iterations was set to 10^{-7}.

7.4 Evaluation Measures

We evaluate each method by comparing the top-k influencers yielded by that method against the top-k

influencers who have the maximum citations. The number of citations received by an author is an indication of the influence the author has and hence we choose this is the ground truth. For this purpose we obtained the citation counts for authors in DBLP and US Patent datasets from [32] and [33] respectively.

We have used standard information retrieval measures such as *Precision, Recall* and F_1*-Measure* to compare the top-k authors returned by a method to the top-k most cited authors. Hence, the top-k most cited authors serve as the relevant set of results (or the ground truth), typically in a web search evaluation scenario, while the top-k author list from each method acts as the retrieved list. Given the relevant and retrieved lists we can compute the evaluation measures. These measures are listed below.

Precision: Simply put, precision measures the fraction of authors retrieved that are relevant (i.e. most cited). Mathematically it can be written as following.

$$precision = \frac{|relevant \cap retrieved|}{|retrieved|}$$

Recall: Recall measures the fraction of relevant authors that are actually retrieved. Mathematically it can be formulated as below.

$$recall = \frac{|relevant \cap retrieved|}{|relevant|}$$

F_1-Measure: F_1-Measure is the harmonic mean of Precision and Recall measures and factors each of them evenly to give a holistic measure of the method's accuracy.

$$F_1 - Measure = \frac{2 \times precision \times recall}{precision + recall}$$

7.5 Effectiveness Analysis

We do the effectiveness analysis over two sets of graphs for the DBLP and US Patent data. The first set is weighed using the simple graph-weighing scheme explained in Section 6.2 and the second set if weighed using Newman's hypergraph based weighing scheme explained in section 6.5.3. We use Newman's scheme since we have established that it's the most appropriate hyperedge weighing scheme in section 6.6.

We study the effectiveness of our proposed approach to the baselines by computing the evaluation measures stated in the previous section (section 7.4) for our method as well as the baseline for varying values of top-K from 1 to 500 and measuring the average performance of each of the methods over this varying top-k value.

7.5.1 Weighted Graph

For this effectiveness analysis we used simple graph weighing measure explained in section 6.2 to create the DBLP and SP Patent graphs. We measured the precision for the DBLP dataset with varying top-k values in Figure 7.2(a). As one can see our proposed approach *SoCap* finds up-to *16%* of the 500 most cited authors, while the best baseline method PageRank can only find *8%*. We have also measured the average precision over the top-500 authors for each method, which is shown in braces in the figure. The average score indicates the overall performance of each method measured over the entire range of top-k = 1 to top-k = 500 list. Our method has around *4%* higher average precision as compared to the best baseline. Our method also performs the best in terms of recall and F_1-Measure as shown in Figure 7.2(b) and 7.2(d), better than the best baseline by up-to *8%*. This consistently shows up

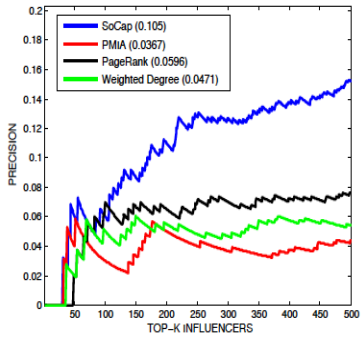
in the precision-recall curve where our method is consistently above all the other baselines.

When the same set of experiments were performed on the US Patent dataset our method was still superior but by a lesser margin. We hypothesize that this is due to the extremely sparse nature of the UP Patent dataset, which is evident from the number of weakly connected components 417,933 that is 4X larger than the DBLP network. This is because collaborations amongst inventors are more closed as compared to academic publications, and rarely there are inventions spanning across multiple organizations or groups. Due to the intermingled nature of the curves, we use average measure as the indicator of overall performance and we can see in Figures 7.2(e), 7.2(f) and 7.2(h) that our method beats all the baselines on the average.

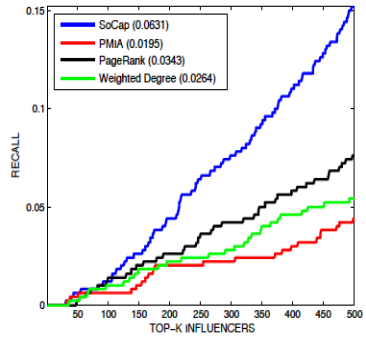
7.5.2 WEIGHTED HYPERGRAPH

When hypergraph weighing scheme is used in place of simple graph weighing scheme the results appear pretty similar through the Precision, Recall and F_1-Measure curves for both the DBLP and US Patent datasets. *SoCap(HG)* is our algorithm using the hypergraph weighing

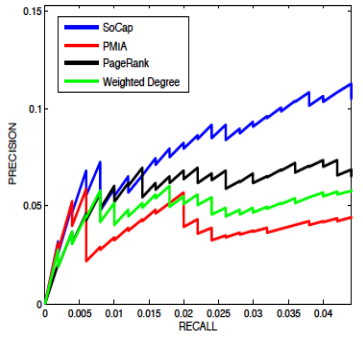
scheme. The effectiveness curves are shown in Figure 7.3 below.



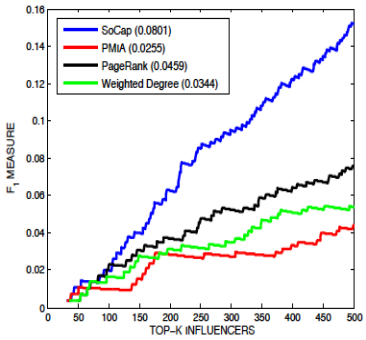
(a) Precision measured for top-k influencers for DBLP



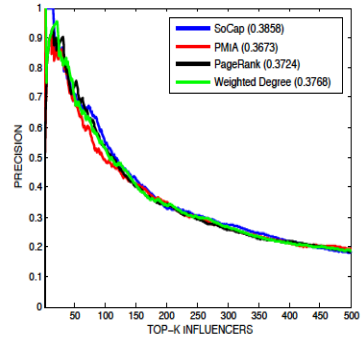
(b) Recall measured for top-k influencers for DBLP



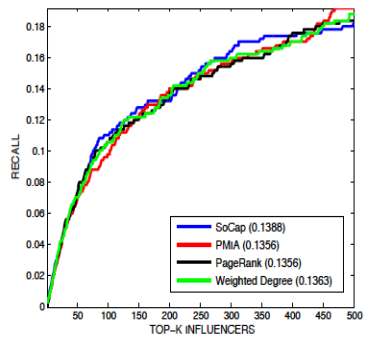
(c) Precision-Recall curve for DBLP



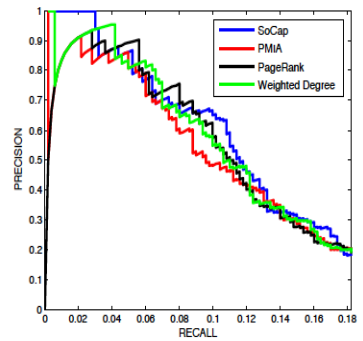
(d) F_1-measure for DBLP



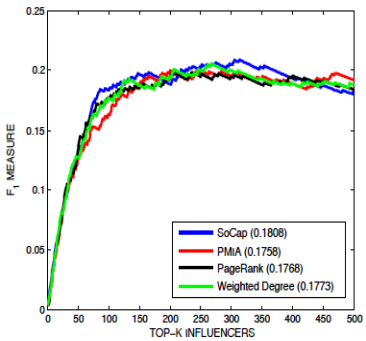
(e) Precision measured for top-k influencers for USPTO



(f) Recall measured for top-k influencers for USPTO

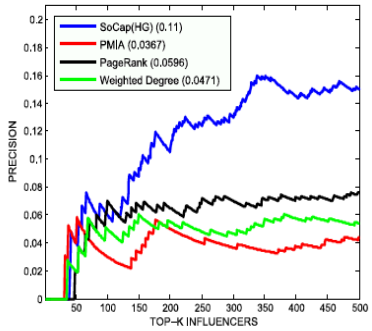


(g) Precision-Recall curve for USPTO

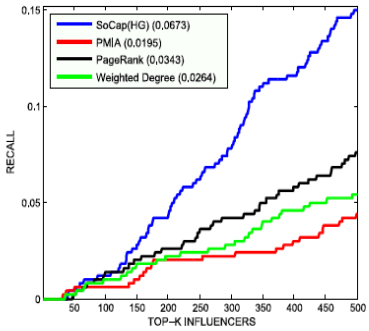


(h) F_1-measure for USPTO

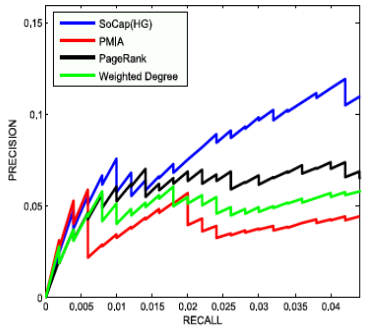
Figure 7.2: The effectiveness results for DBLP and US Patent datasets are shown for tok-k influencers compared against top-k most cited authors for graph weighing scheme



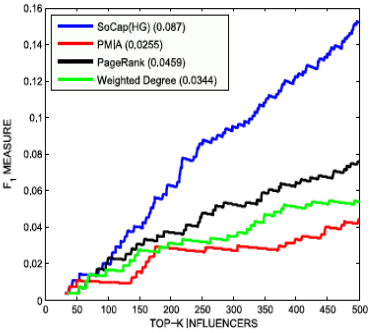
(a) Precision measured for top-k influencers for DBLP



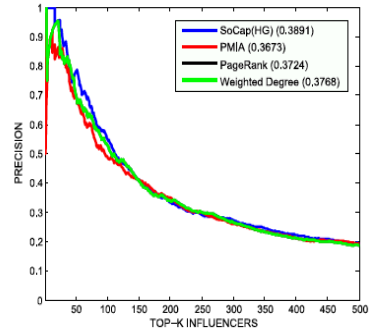
(b) Recall measured for top-k influencers for DBLP



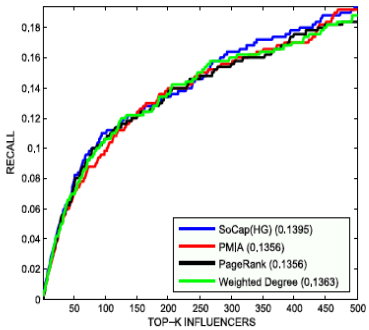
(c) Precision-Recall curve for DBLP



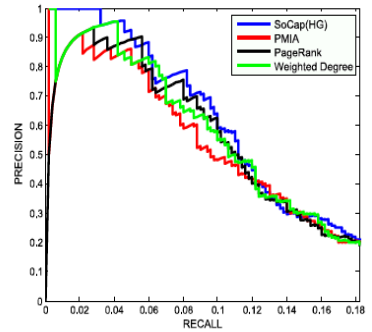
(d) F_1-measure for DBLP



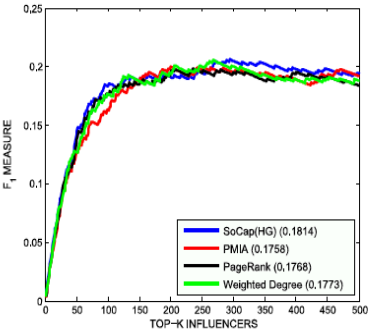
(e) Precision measured for top-k influencers for USPTO



(f) Recall measured for top-k influencers for USPTO



(g) Precision-Recall curve for USPTO



(h) F_1-measure for USPTO

Figure 7.3: The effectiveness results for DBLP and US Patent datasets are shown for tok-k influencers compared against top-k most cited authors using Hypergraph weighing scheme

7.5.3 Graphs v/s Hypergraphs

Even though the overall Precision, Recall and F_1-Measure curves look similar we can observe that on an average hypergraph weighing scheme does better than the simple graph weighing scheme, and hence we can conclude that hypergraphs are better tools for modeling group level interactions as they capture extra information which is missed out by graph modeling. The average numbers for both of these schemes are given below in Table 7.2. We also performed a t-test on the values used for computing the averages and the difference between the two methods was ststistically significant at 5% significance level.

	DBLP		**US Patent**	
Measure	*SoCap*	*SoCap(HG)*	*SoCap*	*SoCap(HG)*
Precision	0.1050	0.1101	0.3858	0.3891
Recall	0.0631	0.0673	0.1388	0.1395
F_1-Measure	0.0801	0.0870	0.1808	0.1814

Table 7.2: Average Precision, Recall and F_1-Measure for DBLP and US Patent datasets compared against two variants of our proposed approach SoCap and SoCap(HG)

7.6 Case Study

In this section we discuss a case study of the top 10 most cited authors in the DBLP and the US Patent dataset, and their ranks as determined by various methods (including our method). Top 10 most cited authors of the DBLP and US Patent datasets are listed in Table 7.3 and Table 7.4 respectively. We list the ranks of these influencers as found in the top-1000 list generated by each mehtod. And, if the influencer is missing in the top-1000 list of a method it is denoted by a '-' in that column. The influencers are ranked in each method from 1 (top influencer) to 1000.

Our methods *SoCap* and *SoCap(HG)* perform extremenly well for the DBLP network since there is strong academic collaboration forming high bonding and bridging nodes. As shown in Table 7.3 both of our methods found 9 out of the top 10 influencers in the top 1000 list and always in a better position if that influencer is found by a baseline.

In the US Patent network we find that George Spector is consistently found in the top 2 positions in all the methods. But, in general the baseline methods as well as our simple graph weighing scheme approach (*SoCap*) nearly

missed 50% of the listed influencers. Howerer, the hypergraph based approach identifies *70%* of the listed top influencers. The high ratio of misses is most likely due to the highly disconnected nature of the US Patent network.

However, for both the datasets we can see that our methods consistently find the listed influecners in higher ranks whenever the baseline also finds the influencer. Hence, this case study also establishes the superiority of our proposed approach as compared to the baselines.

Influencer/Method	SocCap	SocCap (HG)	Page Rank	Wtd. Deg.	PMIA
Jeffery D. Ullman	753	930	-	-	-
Rakesh Agrawal	492	439	-	-	-
Hector Garcia-molina	218	275	399	541	832
David S. Johnson	-	-	-	-	-
Jiawei Han	44	54	158	219	447
Scott Shanker	445	294	-	-	-
Christos Faloutsos	92	60	225	328	719
David E. Culler	450	585	-	-	-
David J. Dewitt	416	460	-	-	-
Hari Balakrishnan	866	851	-	-	-

Talbe 7.3: Case study of top 10 most cited authors in DBLP dataset and their ranks (if < 1000) in various methods. '-' denotes that the author was not found in the top 1000 of the method

Influencer/Method	SoCap	SoCap (Hg)	Page Rank	Wtd. Deg.	PMIA
Felix Theeuwes	-	734	-	-	-
Roshantha Chandra	-	-	-	-	-
Shunpei Yamazaki	107	264	400	546	-
Donald E. Weder	-	149	-	183	139
Kary B. Mullis	-	-	-	-	-
Yasushi Sato	160	121	206	258	429
George Spector	1	1	2	1	1
Jerome Lemelson	493	512	-	776	542
Charles Ichelberger	-	-	-	-	-
Terry M. Haber	621	628	-	-	709

Talbe 7.4: Case study of top 10 most cited authors in US Patent dataset and their ranks (if < 1000) in various methods. '-' denotes that the author was not found in the top 1000 of the method

7.7 Efficiency Analysis

We compare the run times of baseline algorithms against our proposed approach in Table 7.5 below. All the experiments were performed on a Windows 2008 Server, having two Intel Xeon processors of 2.67 GHz, with 8GB RAM. The code was implemented using Java 1.6.

Method	DBLP	US Patent
PMIA	4233.76	3356.04
PageRank	1001.43	856.67
Weighted Degree	352.21	239.75
SoCap & SoCap(HG)	3382.29	1372.52

Table 7.5: The run time reported in seconds for baseline and proposed approach

The PageRank and Weighted Degree algorithms are fast but not as effective as our approach SoCap (and SoCap(HG)) as shown in Figure 7.2 (and Figure 7.3). Our proposed approach takes similar time as PageRank in sparse US Patent network while giving a higher average performance in terms of Precision, Recall and F_1-Measure. While in the reletively denser (yet sparse) DBLP network our approach works extremely well of upto 8% on precision, recall and F_1-Measure. However, it takes more time as compared to centrality measures like Weighted Degree and PageRank. PMIA is both compute intensive and not as efficient as our proposed approach.

Chapter 8

Distributed Paradigms Discussion

8.1 Introduction

There have been recent advances in technologies that now enable us to generate, store and analyze huge volumes of data; also collectively referred to as “big data” technologies. This chapter discusses a couple of popular distributed frameworks, which could be utilized to implement the proposed method so that it can handle graphs of such huge scale. The frameworks that we will discuss in this chapter are the MapReduce framework and Gather Apply Scatter (GAS) programming framework. These frameworks are meant for data, which cannot fit into the memory of a single machine and uses commodity hardware in order to give large storage, computational power and configurability. We try to give an outline of the advantages and limitations of each of these with respect to the implementation of our method.

8.2 The MapReduce Framework

The MapReduce framework is an extremely distributed programming paradigm. We have used Apache Hadoop [40] as it is an open source version which is based upon the MapReduce paradigm. There are two important components of Hadoop, one is HDFS (Hadoop Distributed File System) and the other is the MapReduce programming framework. Both of them are closely coupled since each node in the cluster acts in multiple capacities like the Name node, Data node, Job Tracker or Task Tracker. Hadoop automatically takes the advantage of locality of data and executes parallel jobs in such a way that minimal data exchange is required, and that too exchange between closer (in terms of locality/bandwidth) servers is preferred. From a very high level perspective Hadoop can be thought of as a sorting and shuffling framework for extremely large data. They key notions for programming on MapReduce are keys and values, and MapReduce sorts the data by keys and then does the user defined aggregation on the values of the same key and gives a result. Hence, Hadoop is extremely useful for doing aggregation operations for huge datasets.

Iterative graph algorithms like the proposed method can also be implemented using Hadoop and of course in an extremely scalable manner. But, there are a couple of key challenges which need to be carefully thought about. The first one is that Hadoop is not intuitive for iterative graph algorithms. Since, the key unit of thinking for MapReduce implementations is key value pairs therefore it is not very straightforward to think about iterative graph algorithms for such an implementation. The second challenge is that after each iteration a mapper task and a reducer task both read and write data to the disk. A typical graph algorithm needs to maintain the state of its nodes across several iterations. However the state of very few nodes is updated in each iteration. If each iteration of a graph algorithm is mapped onto a single map-reduce task, then for each iteration the state of the entire graph will be read and written to the disk. This makes the I/O costs increase significantly. Another alternative framework which suits iterative graph algorithms better is the GAS framework which is discussed in the section below.

8.3 The GAS Framework

The MapReduce framework, in spite of being scalable is not ideally suited for iterative operations on graphs.

The primary reason is that each iteration involves reading and writing the whole graph from and to the disk. Since the problem we are tackling involves large social graphs, this drawback is very serious and needs a workaround. The Gather-Apply-Scatter model of computation is vertex centric and this is how it overcomes the drawbacks of the MapReduce paradigm. During the course of this section, we will discuss details about the Apache Giraph [41] implementation of the Gather-Apply-Scatter model. Bulk Synchronous Parallel (BSP) was proposed by Leslie Valiant in 1990 [42]. The BSP model comprises of components which perform processing and other memory functions, routers or other communication media which delivers and facilitates the exchange of messages amongst the components and a certain piece to facilitate and allow synchronization at regular time intervals. Computation in a BSP model consists of numerous supersteps where each superstep consists of a computation phase where each processor makes use of only the locally held values and a mechanism of global message transmission from each processor to any subset of others and towards the end of the superstep, a feature of barrier synchronization. Any exchange of messages done in the immediate previous superstep is available as local data in the current superstep to be executed. After a certain preset time, a global check is made to ascertain

if or not each processor has completed its allotted task. If so, the computation proceeds to next superstep. If not time is allocated to the unfinished supersteps on other machines. There are several GAS frameworks available which do not necessarily follow this paradigm.

For Giraph, the crux of the framework is the Compute method. Every active vertex calls the Compute method exactly once during a superstep. The messages received in the previous superstep are processed in the current superstep inside the compute method. A vertex is active in the begening of the superstep if it has a non empty message queue after the last superstep, otherwise the vertex simply is in a halted state and won't call the compute method till it becomes active at the start of some superstep. It's important to note here that at the zeroth superstep all the vertices are in the active state.

This method is quite similar to the mapper in Hadoop. The users provide their own implementation of the Compute method for a vertex which inturn is of the format – I – Vertex ID; V – Vertex Value; E – Edge Value and M – Message Value.

Chapter 9

Conclusions and Future Work

9.1 Conclusions

The problem of identifying key influencers in networks is critical for many domains from marketing to public health. This problem has continually held interest of these communities and more effective and efficient approaches are greatly beneficial.

In this work we have devised a new approach towards calculating (social capital) value of nodes in a given network, and have used social capital to identify the key influencers in a network. This approach is based on the popular valuation-allocation model, wherein we calculate the value for the entire network and divide it fairly amongst the participating nodes (this is their social capital value). The fairness is ensured by our allocation function which falls in the class of Myerson's allocation functions.

Also, we developed extremely efficient sparse graph algorithms to implement our shortest path based allocation function for both weighted and un-weighted graphs. We empirically established that the effectiveness obtained using the proposed approach was superior to the other popular baseline approaches on two real life datasets. We showed that for extremely sparse networks our algorithm is computationally efficient and gives a better performance on the average as compared to the baselines. And, on relatively denser networks our algorithm shows an improvement of up to *8%* in terms of our evaluation measures (precision, recall and F_1-Measure) as compared to the best baseline method.

Moreover, we used hypergraphs, which are a generalization of graphs as a modeling tool for the given datasets. We tried using five different weighing schemes for measuring the strength of ties of a group (hyperedge) and empirically identified Newman's definition to be consistently better as compared to the others on two collaborative datasets, one from the real world and the other from the virtual world.

Furthermore, we devised an algorithm to condense a given hypergraph into a simple weighted graph (which captures

more information), Which enables existing graph algorithms to be applicable. We also empirically established that using hypergraphs for modeling group interaction networks gave a better performance as compared to the corresponding simple graph modeling on an average.

Finally, we explored (in theory) two popular distributed paradigms MapReduce and Bulk Synchronous Parallel for the extension of our algorithm to extremely large scale graphs.

9.2 Future Work

One can further extend our model by incorporating temporal weighing schemes to capture dynamics of temporally evolving networks by capturing the decaying strength of ties between people/groups as time passes without any activity.

Also, sophisticated machine learning techniques can be used to learn the various parameters like maximum hop length to consider (hop_{max}) and value of the decay constant (λ) that are provided to the algorithm.

Furthermore, with the rapid increase in the volume and the velocity at which data is being generated graphs are quickly scaling to unimaginable sizes (billion nodes plus). Single core algorithms for such large-scale graphs are going to quickly become obsolete. Our method can be easily be implemented on top of distributed paradigms like MapReduce and Bulk Synchronous Parallel. These algorithms and frameworks have been discussed in Chapter 8. Our algorithm is a natural fit to the Bulk Synchronous Parallel paradigm and frameworks like Apache Giraph and GraphLab [34], and such an implementation would be a possible direction for testing the extreme scalability of our algorithm.

References

[1] K. Subbian and P. Melville, "Supervised rank aggregation for predicting influencers in twitter, " in SocialCom, 2011, pp. 661-665.

[2] X. Tang and C. C. Yang, "Ranking user influence in healthcare social media, " ACM Transactions on Intelligent Systems and Technology (TIST), vol. 3, no. 4, p. 73, 2012.

[3] J. C. Bertot, P. T. Jaeger, and D. Hansen, "The impact of polices on government social media usage: Issues, challenges, and recommendations, "Government Information Quarterly, vol. 29, no. 1, pp. 30--40, 2012.

[4] D. Kempe, J. M. Kleinberg, and Éva Tardos, "Maximizing the spread of influence through a social network, " in KDD, 2003, pp. 137--146.

[5] W. Chen, C. Wang, and Y. Wang, "Scalable influence maximization for prevalent viral marketing in large-scale social networks, " in KDD, 2010, pp. 1029--1038.

[6] J. Leskovec, A. Krause, C. Guestrin, C. Faloutsos, J. VanBriesen, and N. S. Glance, "Cost-effective outbreak detection in networks, " in KDD, 2007, pp. 420--429.

[7] W. Chen, Y. Wang, and S. Yang, "Efficient influence maximization in social networks, " in KDD , 2009, pp. 199--208.

[8] M. Kimura and K. Saito, "Tractable models for information diffusion in social networks, " in KDD , 2006, pp. 259--271.

[9] K. Subbian, C. Aggarwal, and J. Srivastava, "Content-centric flow mining for influence analysis in social streams, " in CIKM , 2013, pp. 841--846.

[10] R. S. Burt, Brokerage and Closure : An Introduction to Social Capital. Oxford University Press, 2005.

[11] P. Dekker and E. M. Uslaner, Social Capital and Participation in Everyday Life. Rouledge, 2001.

[12] D. Knoke, Organizational Networks and Corporate Social capital. Kluwer, 1999.

[13] M. O. Jackson and A. Wolinsky, "A strategic model of social and economic networks, " in Jour. of Eco. Theo. , vol. 71, no. 1, 1996, pp. 44--74.

[14] R. B. Myerson, "Graphs and cooperation in games, " in Mathematics of Operations Research , vol. 2, no. 3, 1977, pp. 225--229.

[15] P. Melville, K. Subbian, C. Perlich, R. Lawrence, and E. Meliksetian, "A predictive perspective on measures of influence in networks, " in Proceedings of the Workshop on Information in Networks , 2010.

[16] S. Goyal and F. Vega-Redondo, "Structural holes in social networks, " in Journal of Economic Thoery , vol. 137, no. 1, 2007, pp. 460--492.

[17] J. Kleinberg, S. Suri, E. Tardos, and T. Wexler, "Strategic network formation with structural holes, " in Elec. Comm. , 2008, pp. 132--141.

[18] R. Narayanam and Y. Narahari, "Topologies of strategically formed social networks based on a generic value function - allocation rule model, " in Social Networks , vol. 33, 2011, pp. 56--69.

[19] M. Smith, C. Giraud-Carrier, and B. Judkins, "Implicit affinity networks, " in Annual Workshop on Information Technologies and Systems , 2007, pp. 8--13.

[20] M. Smith, "Social capital in online communities, " in Workshop on PhD Students in Information and Knowledge Management , 2008, pp. 17--24.

[21] M. Smith, C. Giraud-Carrier, and N. Purser, "Implicity affinity networks and social capital, " in Information Technology and Management , 2009, pp. 123--134.

[22] C. Johnson and R. P. Gilles, "Spatial social networks, " in Rev. of Eco. Des. , vol. 5, no. 3, 2000, pp. 273--299.

[23] B. Dutta, A. V. D. Nouweland, and S. Tijs, "Link formation in cooperative situtations, " in Int.

Jour. of Game Theo. , vol. 27, no. 2, 1998, pp. 245--256.

[24] L. S. Shapley, "A value for n-person games, " In Contributions to the Theory of Games, volume II, by H.W. Kuhn and A.W. Tucker, editors. Annals of Mathematical Studies , vol. 28, pp. 307--317, 1953.

[25] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, Introduction to algorithms. MIT press, 2001.

[26] C. Berge and E. Minieka, Graphs and hypergraphs. North-Holland publishing company Amsterdam, 1973, vol. 7.

[27] E. Estrada and J. A. Rodriguez-Velazquez, "Complex networks as hypergraphs, " arXiv preprint physics/0505137 , 2005.

[28] J.-W. Wang, L.-L. Rong, Q.-H. Deng, and J.-Y. Zhang, "Evolving hypernetwork model, " The European Physical Journal B , vol. 77, no. 4, pp. 493--498, 2010.

[29] K. Kapoor, D. Sharma, and J. Srivastava, "Weighted node degree centrality for hypergraphs, " in Network

Science Workshop (NSW), 2013 IEEE 2nd. IEEE, 2013, pp. 152--155.

[30] M. E. Newman, "Scientific collaboration networks. ii. shortest paths, weighted networks, and centrality, " Physical review E , vol. 64, no. 1, p. 016132, 2001.

[31] J. Gao, Q. Zhao, W. Ren, A. Swami, R. Ramanathan, and A. Bar-Noy, "Dynamic shortest path algorithms for hypergraphs, " in Modeling and Optimization in Mobile, Ad Hoc and Wireless Networks (WiOpt), 2012 10th International Symposium on}. IEEE, 2012, pp. 238--245.

[32] Y. Li, B. Liu, and S. Sarawagi, Eds., Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Las Vegas, Nevada, USA, August 24-27, 2008. ACM, 2008.

[33] http://www.google.com/googlebooks/uspto.html.

[34] Y. Low, D. Bickson, J. Gonzalez, C. Guestrin, A. Kyrola, and J. M. Hellerstein, "Distributed graphlab: A framework for machine learning and data mining in

the cloud, " Proceedings of the VLDB Endowment , vol. 5, no. 8, pp. 716--727, 2012.

[35] S. Klamt, U. Haus, and F. Theis, “Hypergraphs and cellular networks,” PLoS computational biology, vol. 5, no. 5, p. e1000385, 2009.

[36] R. Fagin, “Degrees of acyclicity for hypergraphs and relational database schemes,” Journal of the ACM (JACM), vol. 30, no. 3, pp. 514–550, 1983.

[37] E. Han, G. Karypis, V. Kumar, and B. Mobasher, Clustering based on association rule hypergraphs. University of Minnesota, Department of Computer Science, 1997.

[38] D. Zhou and J. Huang, “Learning with hypergraphs: Clustering, classification, and embedding,” 2007.

[39] J. Wang, L. Rong, Q. Deng, and J. Zhang, “Evolving hypernetwork model,” The European Physical Journal B-Condensed Matter and Complex Systems, vol. 77, no. 4, pp. 493–498, 2010.

[40] http://hadoop.apache.org

[41] https://giraph.apache.org

[42] Leslie G. Valiant, “A bridging model for parallel computation, “ Communications of the ACM, Volume 33 Issue 8, Aug. 1990.